Hierarchical-Alphabet Automata and Applications

VUB
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Introduction

- **Who am I?**
  - PhD student at the AI Lab (Applied Research)
  - Under supervision of prof. Johan Loeckx
  - Research interests in applied AI and cybersecurity

- **What is this guest lecture about?**
  - Variants of finite state machines and applications
  - Our hierarchical extension on finite state machines

VUB ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Outline of the lecture

- **Preliminary knowledge:**
  - Terminology + basics of automata theory
  - Tries, factor automata, and factor oracles

- **Our research at the AI lab:**
  - Hierarchical-alphabet automata (HAAs)
  - Hierarchical factor oracles (HFOs)

- **Applications of our research:**
  - Anomaly detection with the HFO

Outline of the lecture

- **Preliminary knowledge:**
  - Terminology + basics of automata theory
  - Tries, factor automata, and factor oracles

- **Our research at the AI lab:**
  - Hierarchical-alphabet automata (HAAs)
  - Hierarchical factor oracles (HFOs)

- **Applications of our research:**
  - Anomaly detection with the HFO

VUB ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Terminology

- **Alphabet:** finite set of symbols
- **Words:** concatenations of zero or more symbols

- **Factors:** contiguous subsequences of a word
  - **Prefix:** a factor at the beginning of a word
  - **Suffix:** a factor at the end of a word

- **Language:** subset of the infinite set of possible words we can create using an alphabet!

Terminology: Examples

- **Alphabet:** { a, b, c, …, 3, 4 }
- **Words:** { "", "a", "abc", "cars", "3x4mpl3", … }

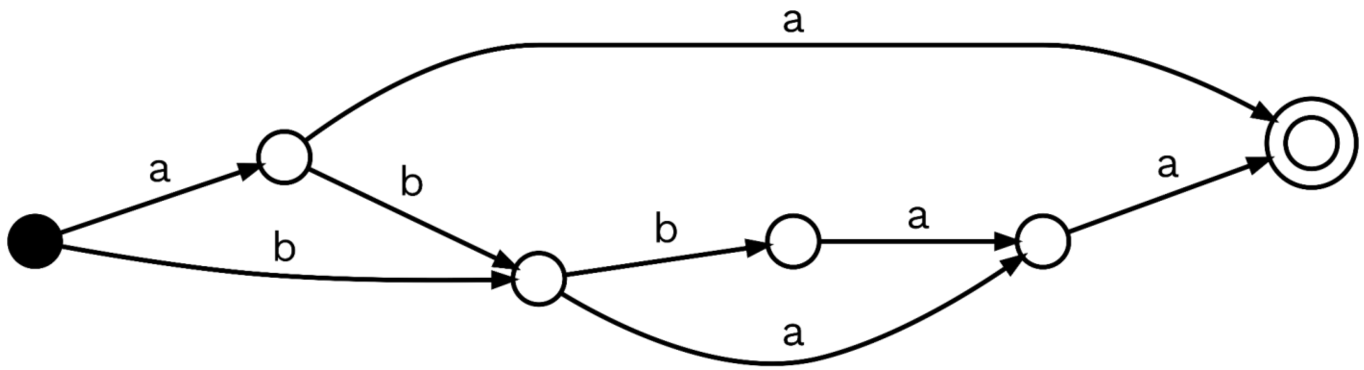
- **Factors:** <u>**actor**</u> is a factor of <u>**factory**</u>
  - **Prefix:** <u>fact</u> is a factor and prefix of <u>factory</u>
  - **Suffix:** <u>ory</u> is a factor and suffix of <u>factory</u>

- **Languages:** a*b*, the set of factors of a word, …

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

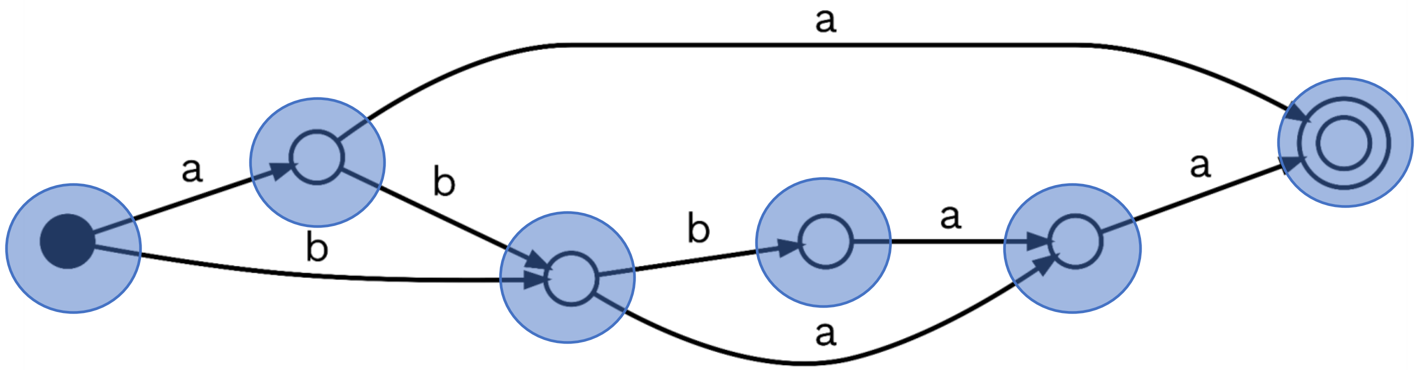
Finite State Machines

- **Definition:** a five-tuple (Q, Σ, δ, $q0$, F)
  - A **finite set of states** Q
  - An **alphabet** (or set of symbols) Σ
  - A **transition function** δ, returns a q of Q
  - The **initial state** $q0$
  - A set of **accepting states** F



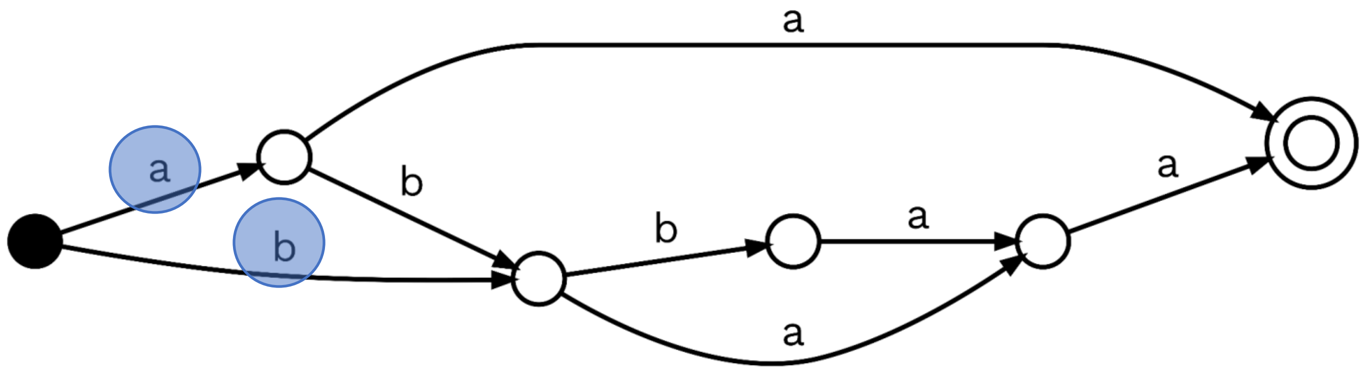
Finite State Machines

- **Definition:** a five-tuple (Q, Σ, δ, $q0$, F)
  - A **finite set of states** Q
  - An **alphabet** (or set of symbols) Σ
  - A **transition function** δ, returns a q of Q
  - The **initial state** $q0$
  - A set of **accepting states** F



Finite State Machines

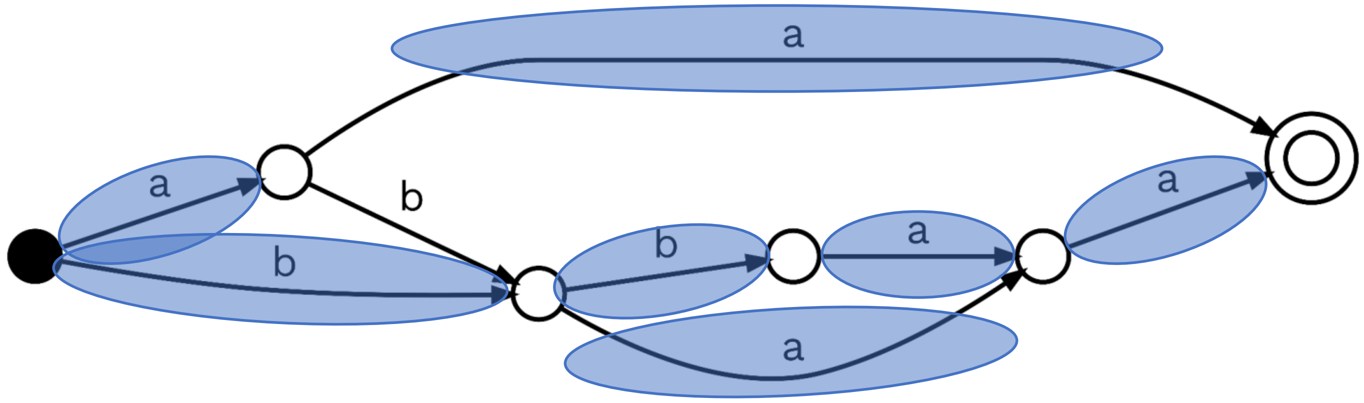
- **Definition:** a five-tuple (Q, Σ, δ, $q0$, F)
  - A **finite set of states** Q
  - An **alphabet** (or set of symbols) Σ
  - A **transition function** δ, returns a q of Q
  - The **initial state** $q0$
  - A set of **accepting states** F



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Finite State Machines

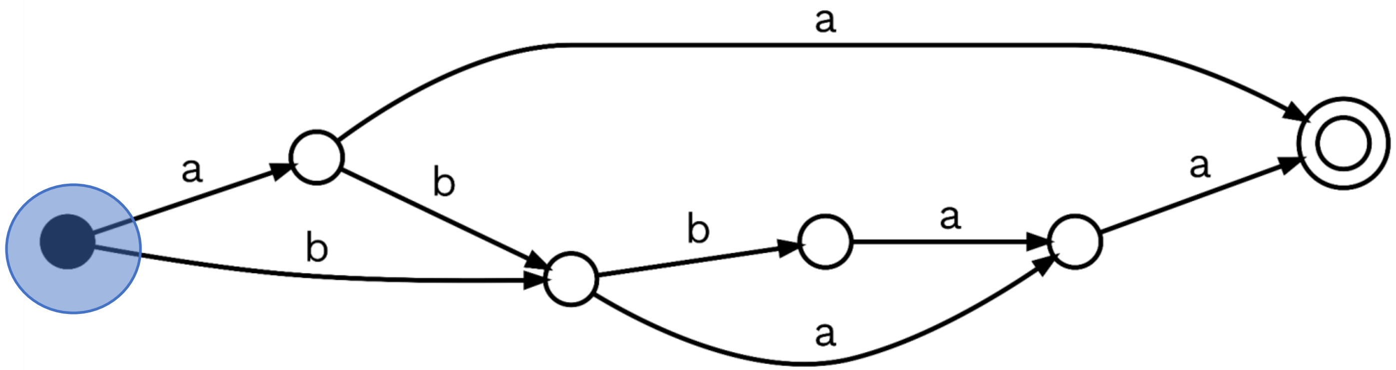
- **Definition:** a five-tuple (Q, Σ, δ, $q0$, F)
  - A **finite set of states** Q
  - An **alphabet** (or set of symbols) Σ
  - A **transition function** δ, returns a q of Q
  - The **initial state** $q0$
  - A set of **accepting states** F



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

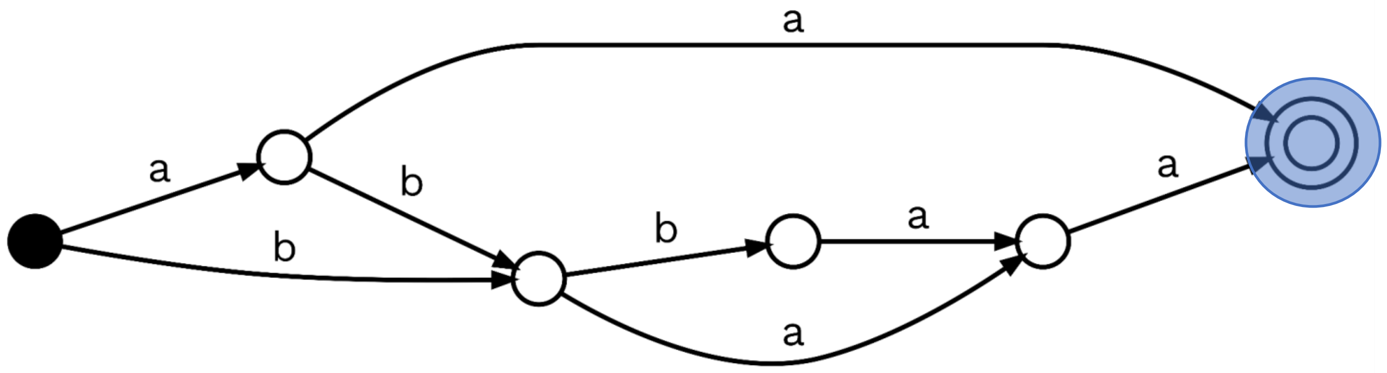
Finite State Machines

- **Definition:** a five-tuple (Q, Σ, δ, $q0$, F)
  - A **finite set of states** Q
  - An **alphabet** (or set of symbols) Σ
  - A **transition function** δ, returns a q of Q
  - The **initial state** $q0$
  - A set of **accepting states** F



Finite State Machines

- **Definition:** a five-tuple $(Q, \Sigma, \delta, q0, F)$
  - A **finite set of states** Q
  - An **alphabet** (or set of symbols) Σ
  - A **transition function** *δ*, returns a *q* of Q
  - The **initial state** *q0*
  - A set of **accepting states** *F*



ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

Finite State Machines

- **Acceptance and rejection of words:**
  - A finite state machine recognizes a **language**:
    - It **accepts** words part of that language
    - It **rejects** words not part of the language
  - Start at the initial state *q0* + first symbol of your input word
  - Repeatedly follow δ with current state + symbol of word:
    - Accepting state after full word? Word is **accepted**
    - Normal state or no transition? Word is **rejected**

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Applications of FSMs

- **Anomaly detection on system call sequences**

```
S0;
while (..) {
      S1;
      if (...) S2;
      else S3;
      if (S4) ... ;
      else S2;
      S5;
}
S3;
S4;
```

$S_0 S_1 S_2$	$S_1 S_2 S_4$	$S_2 S_4 S_5$	$S_3 S_4 S_5$	$S_4 S_5 S_1$	$S_2 S_5 S_1$	$S_5 S_1 S_2$
$S_0 S_1 S_3$	$S_1 S_3 S_4$	$S_2 S_4 S_2$	$S_3 S_4 S_2$	$S_4 S_5 S_3$	$S_2 S_5 S_3$	$S_5 S_1 S_3$
$S_0 S_3 S_4$				$S_4 S_2 S_5$		$S_5 S_3 S_4$

Figure 1. An example program and associated trigrams. S0,...,S5 **denote system calls.**

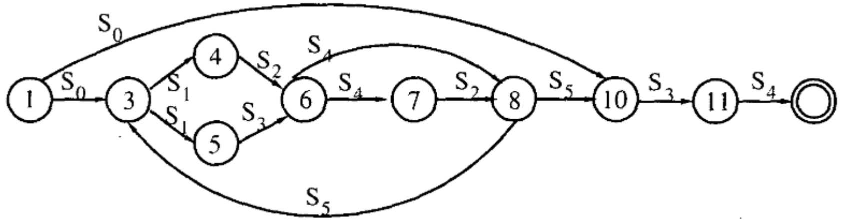


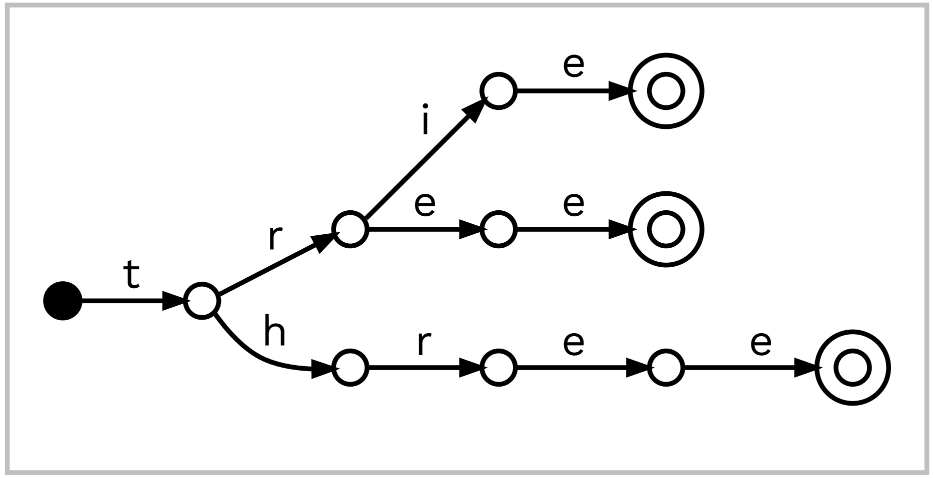
Figure 2. Automaton learnt by our algorithm for Example 1

Sekar, R., Bendre, M., Dhurjati, D., & Bollineni, P. (2000, May). A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001* (pp. 144-155). IEEE.

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Tries

- **Definition:**
  - Rooted tree associated with a set of words
- **Paths from root to leaf represents words from its set**
- **Example:**
  - Trie for the set { *three, tree, trie* }

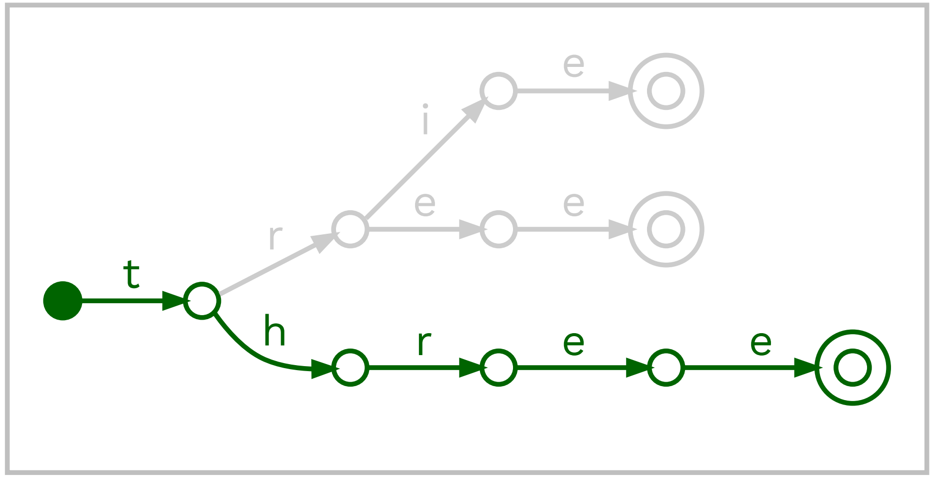


Tries

- **Definition:**
  - Rooted tree associated with a set of words

- **Paths from root to leaf represents words from its set**

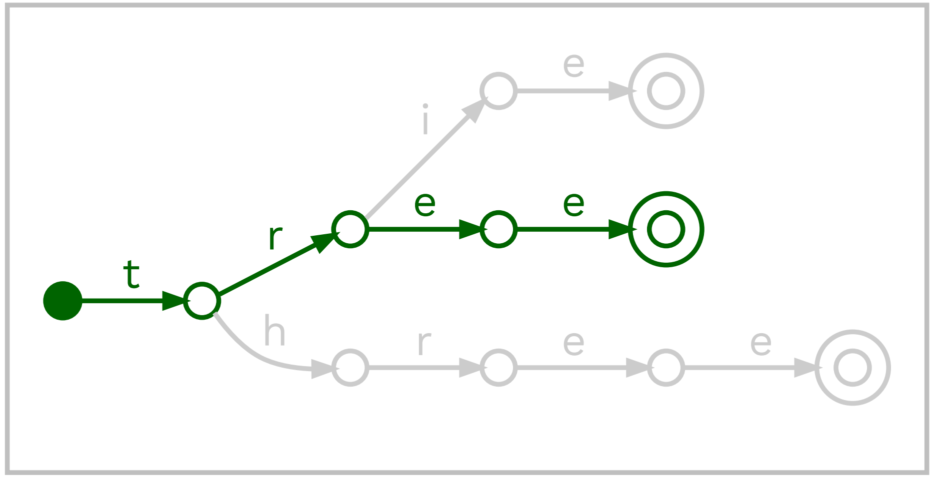
- **Example:**
  - Trie for the set { ***three***, *tree*, *trie* }



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Tries

- **Definition:**
  - Rooted tree associated with a set of words
- **Paths from root to leaf represents words from its set**
- **Example:**
  - Trie for the set { *three*, ***tree***, *trie* }

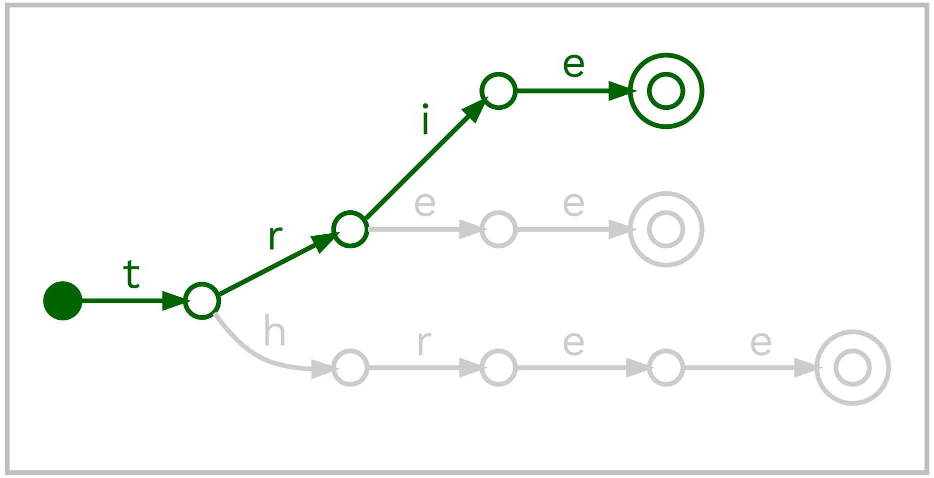


Tries

- **Definition:**
  - Rooted tree associated with a set of words

- **Paths from root to leaf represents words from its set**

- **Example:**
  - Trie for the set { *three, tree, **trie*** }

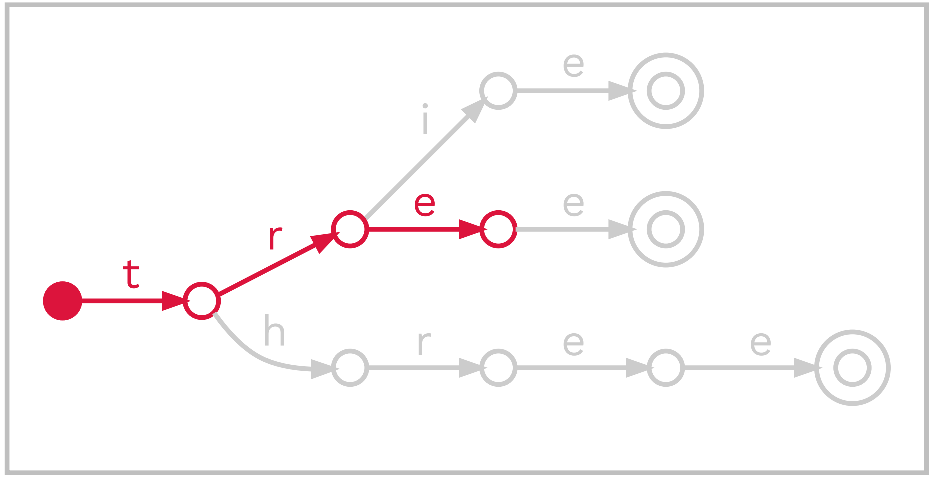


Tries

- **Definition:**
  - Rooted tree associated with a set of words

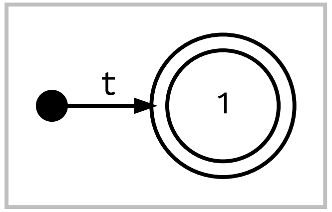
- **Paths from root to leaf represents words from its set**

- **Example:**
  - The word ***tre*** is not in { *three, tree, trie* }



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

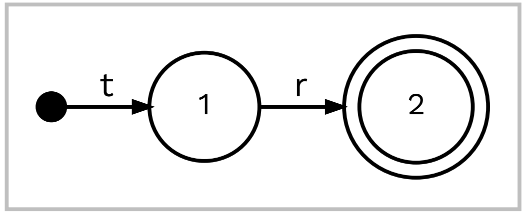
Construction Algorithm of Tries



{**t**rie, tree}

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

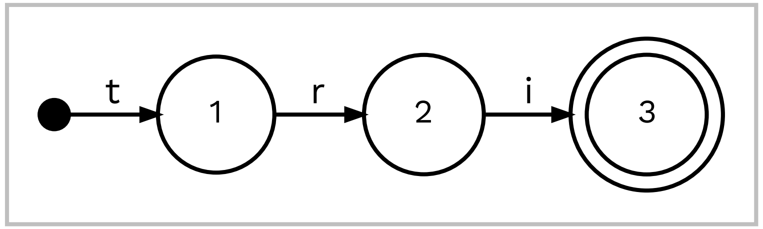
Construction Algorithm of Tries



{**tr**ie, tree}

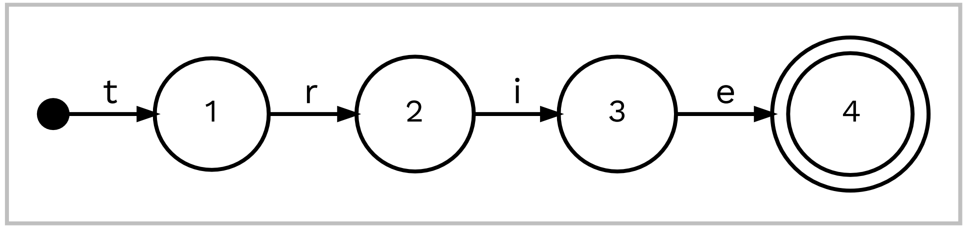
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Construction Algorithm of Tries



{trie, tree}

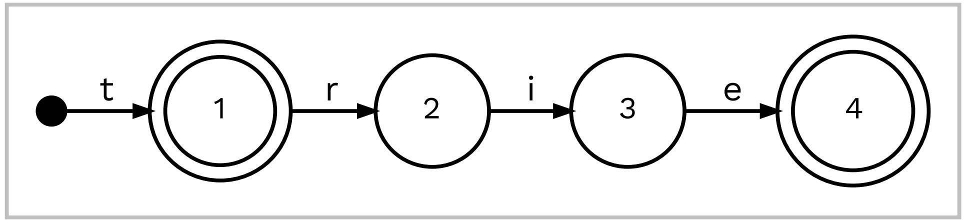
Construction Algorithm of Tries



{**trie**, tree}

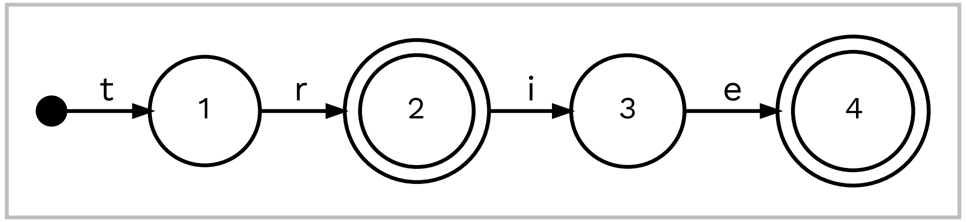
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Construction Algorithm of Tries



{trie, tree}

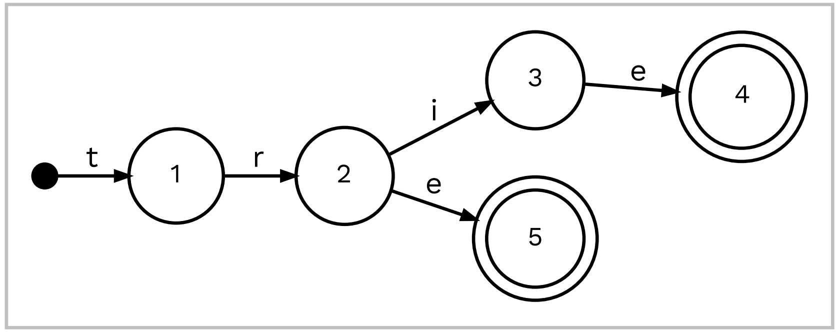
Construction Algorithm of Tries



{trie, tree}

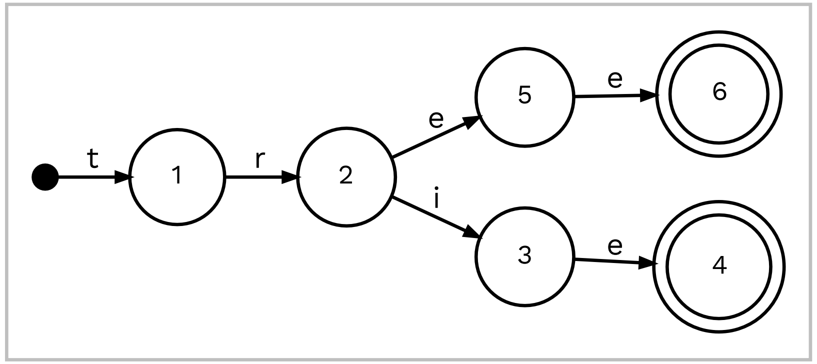
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Construction Algorithm of Tries



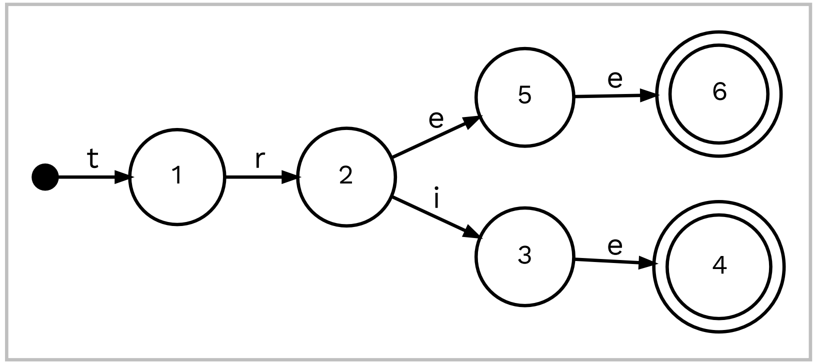
{trie, tree}

Construction Algorithm of Tries



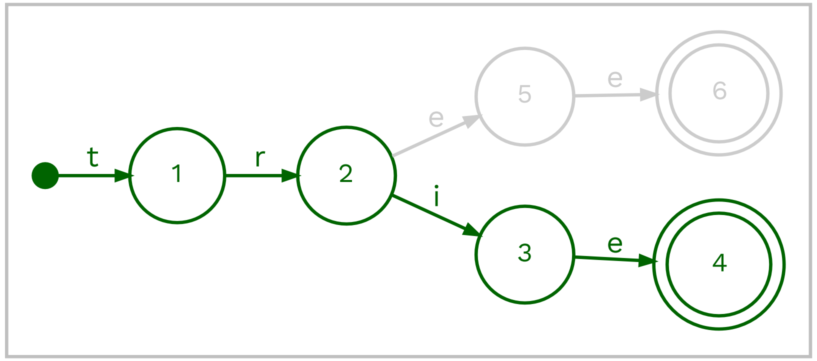
{trie, tree}

Construction Algorithm of Tries



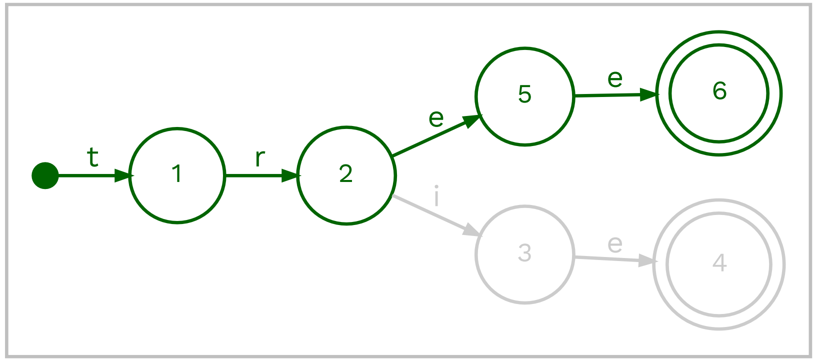
{trie, tree}

Construction Algorithm of Tries



{trie, tree}

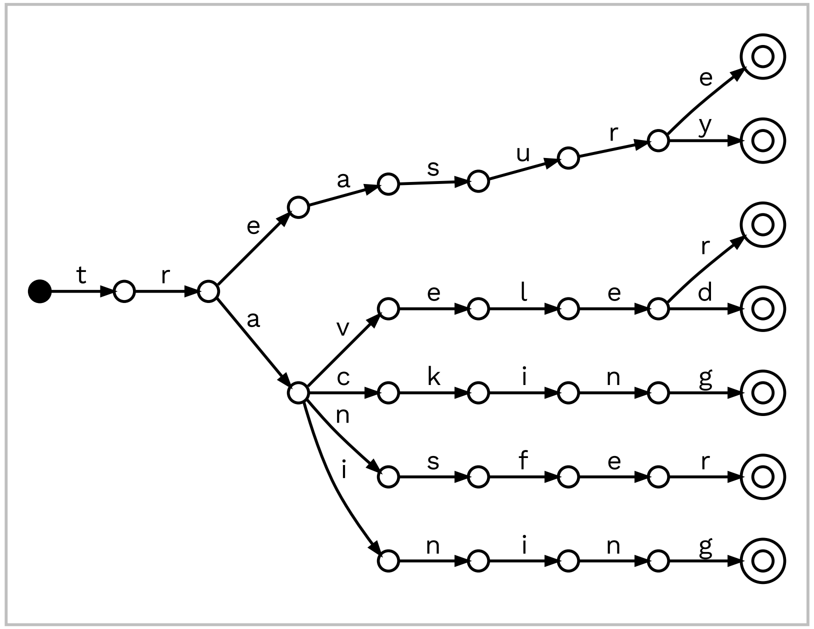
Construction Algorithm of Tries



{trie, tree}

Applications of Tries

- **Efficient data structure, storing words based on prefixes**
  - Eliminates duplicate prefixes, allows for efficient search



Interactive Demonstration of Tries

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

Factor Automata

- **Definition:**
  - Factor automaton of a word *x* is the **minimal** deterministic automaton that **recognizes the factors of *x***

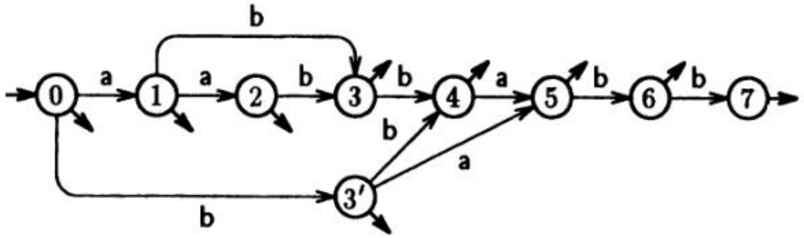


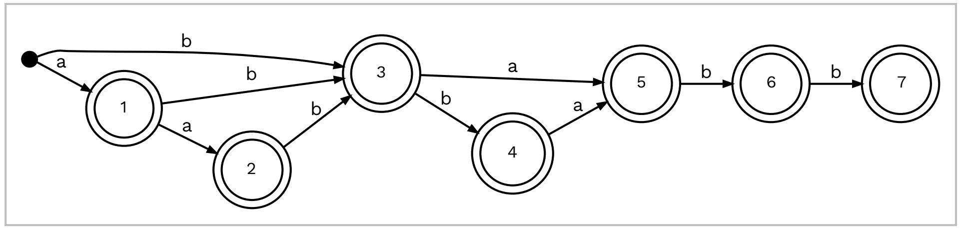
Fig. 7.11. Minimal deterministic automaton recognizing the factors of aabbabb.

Crochemore, M., & Hancart, C. (1997). Automata for matching patterns. *Handbook of Formal Languages: Volume 2. Linear Modeling: Background and Application*, 399-462.

Factor Oracles

- **Definition:**
  - Acyclic automaton built on a (set of) word(s), recognizes **at least** all factors of this (set of) word(s)

- **Applications:**
  - Intended for pattern matching
  - Computation of repeat factors
  - Modelling for music improvisation



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Allauzen, C., Crochemore, M., & Raffinot, M. (1999). Factor oracle: A new structure for pattern matching. In *SOFSEM'99: Theory and Practice of Informatics: 26th Conference on Current Trends in Theory and Practice of Informatics Milovy, Czech Republic, November 27—December 4, 1999 Proceedings 26* (pp. 295-310). Springer Berlin Heidelberg.

Factor Automata vs Factor Oracles

- **Factor automaton:**
  - Deterministic automaton that **recognizes the factors of *x***

- **Factor oracle:**
  - Acyclic automaton that recognizes **at least** all factors of *x*

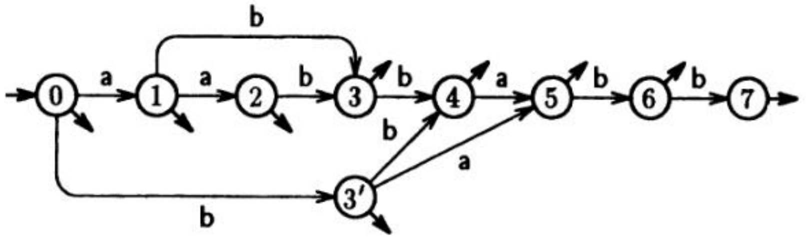
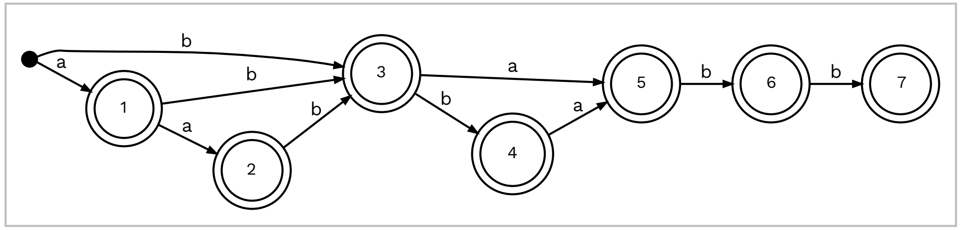


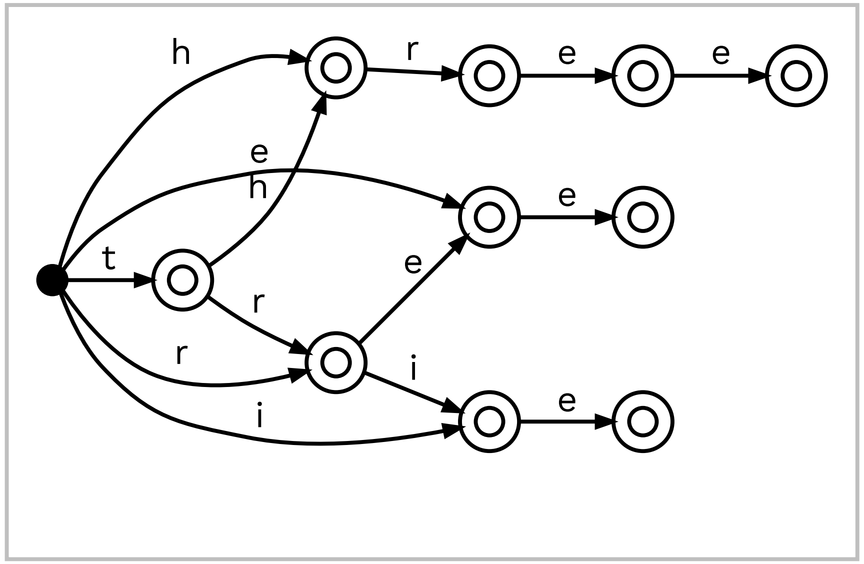
Fig. 7.11. Minimal deterministic automaton recognizing the factors of **aabbabb**.



Factor Automata vs Factor Oracles

- **Factor oracle:**
  - Has an **online algorithm**
  - Constructed in **linear time and space**
  - **Memory-efficient improvement** over FAs

- **Example:**
  - Factor oracle for **{three, trie, tree}**

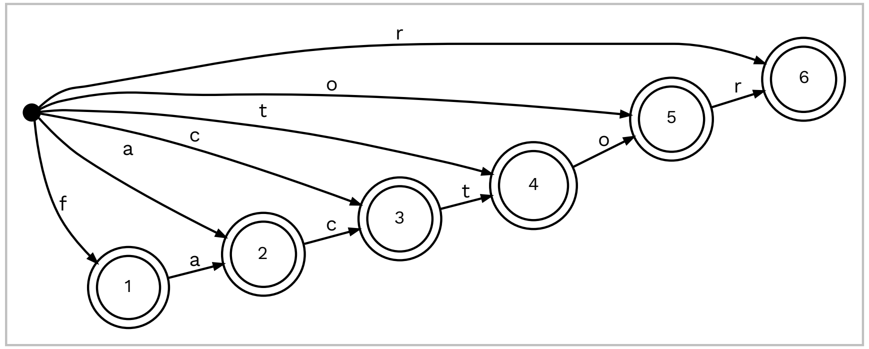


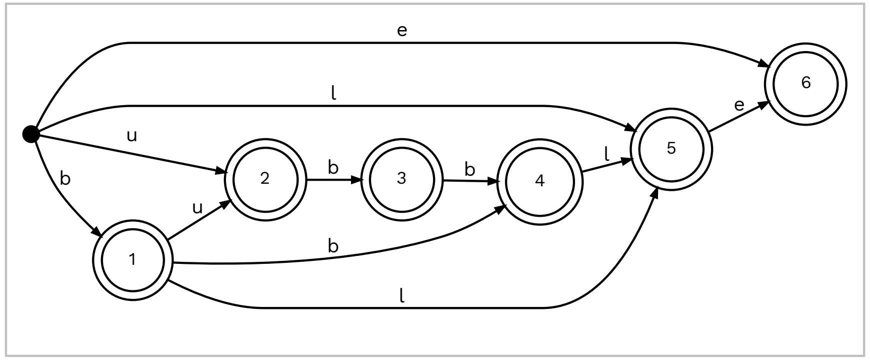
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

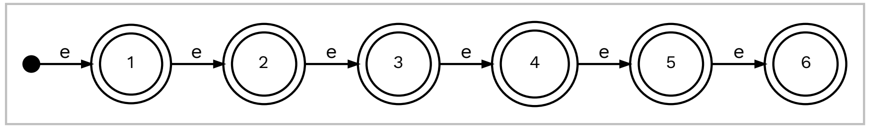
Factor Oracles

- **States and transitions:**
  - Has exactly *m + 1* states
  - Between *m* and *2m – 1* transitions

- **Examples:**
  - FO for the word *"factor"*
  - FO for the word *"bubble"*
  - FO for the word *"eeeeee"*





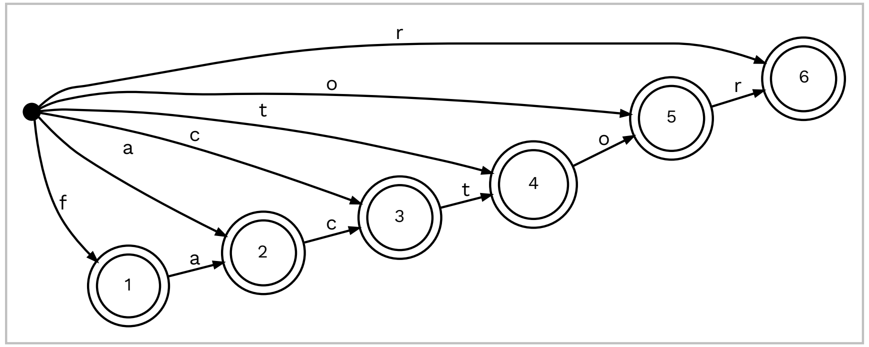


ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Factor Oracles

- **States and transitions:**
  - Has exactly $m + 1$ states
  - Between m and $2m - 1$ transitions

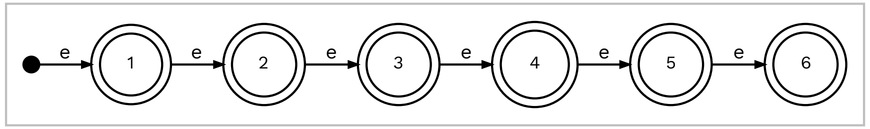
- **Example: "factor"**
  - Word has length 6
  - Has 7 (= 6 + 1) states
  - Has 11 (= 2 * 6 – 1) transitions
  - **Why?** No repeat symbols = **worst-case**



Factor Oracles

- **States and transitions:**
  - Has exactly $m + 1$ states
  - Between m and $2m - 1$ transitions

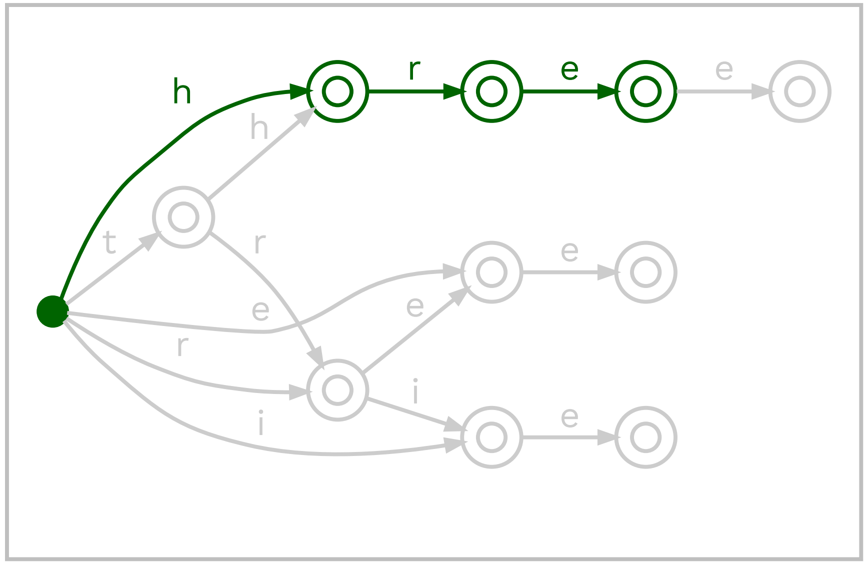
- **Example: "eeeeee"**
  - Word has length 6
  - Has 7 (= 6 + 1) states
  - Has 6 transitions
  - **Why?** Every symbol is equal = **best-case**



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Factor Oracles

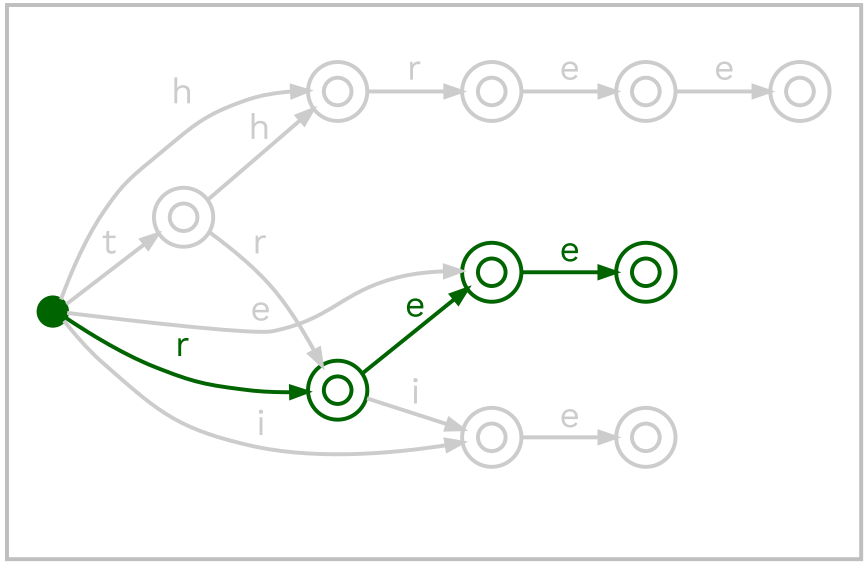
- **Accepts at least all factors:**
  - FO for **{three, trie, tree}**
- **Example:**
  - Accepts "hre", which is a factor of t<u>hre</u>e



ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

Factor Oracles

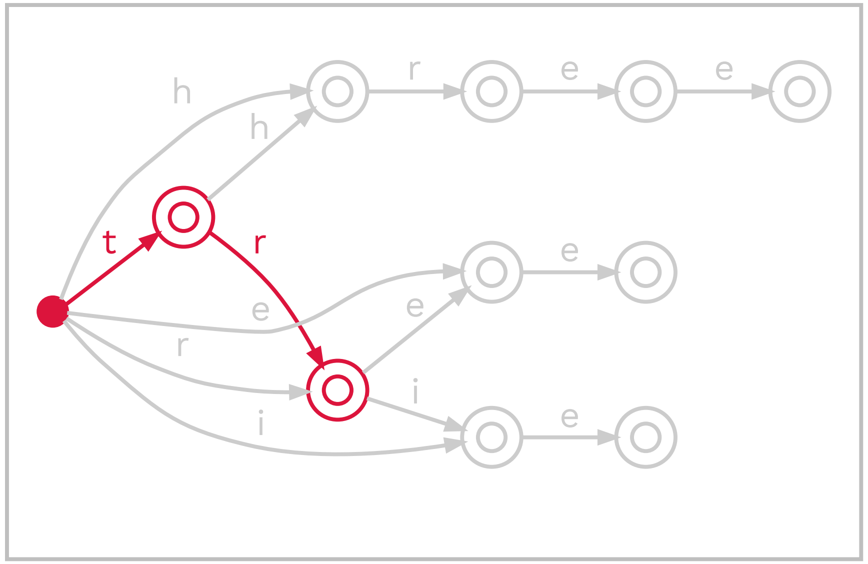
- **Accepts at least all factors:**
  - FO built on {three, trie, tree}
- **Example:**
  - Accepts "ree", which is a factor of both three and tree



Factor Oracles

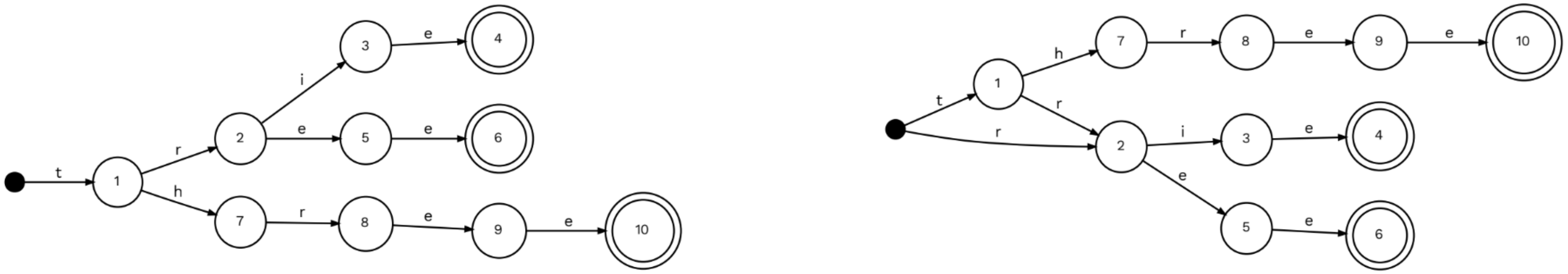
- **Accepts at least all factors:**
  - FO built on {three, trie, tree}

- **Example:**
  - Rejects "trh", because it is not a factor of three, trie, or tree



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Construction Algorithm of FOs

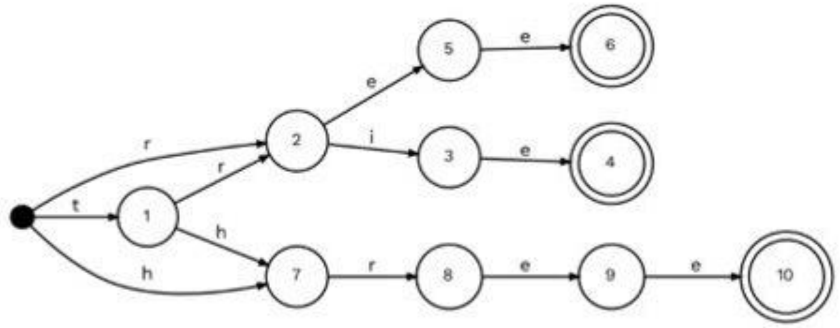


(a) The trie for $P = \{\texttt{trie}, \texttt{tree}, \texttt{three}\}$, taken from *Fig. 2.10*. In the first step of the algorithm, we generate the trie which we use as a starting point to generate the factor oracle.

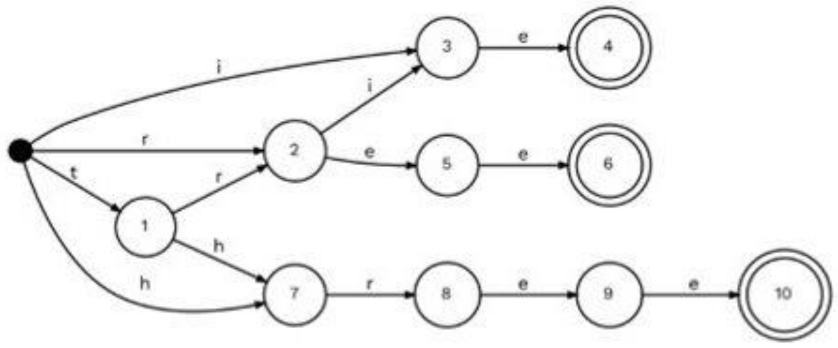
(b) In this step of the construction algorithm, a new transition **from the initial state to state 2** is made, labelled with r. There are now 11 transitions in total.

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Construction Algorithm of FOs



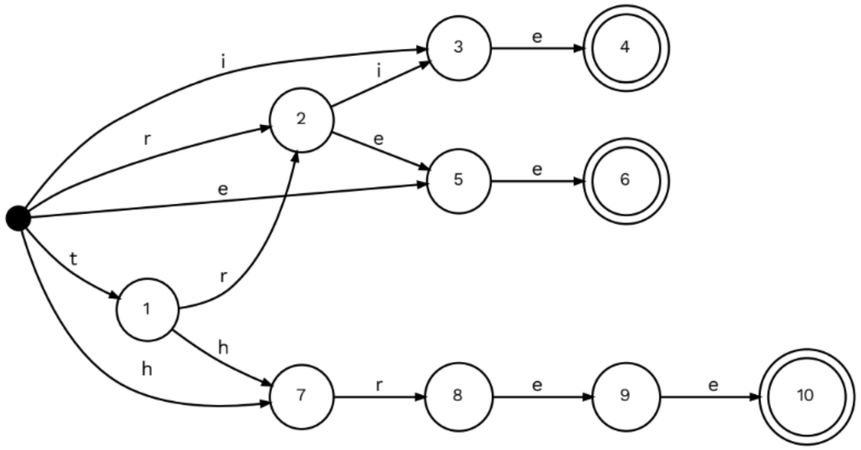
(c) In this step of the construction algorithm, a new transition **from the initial state to state 7** is made, labelled with h. There are now 12 transitions in total.



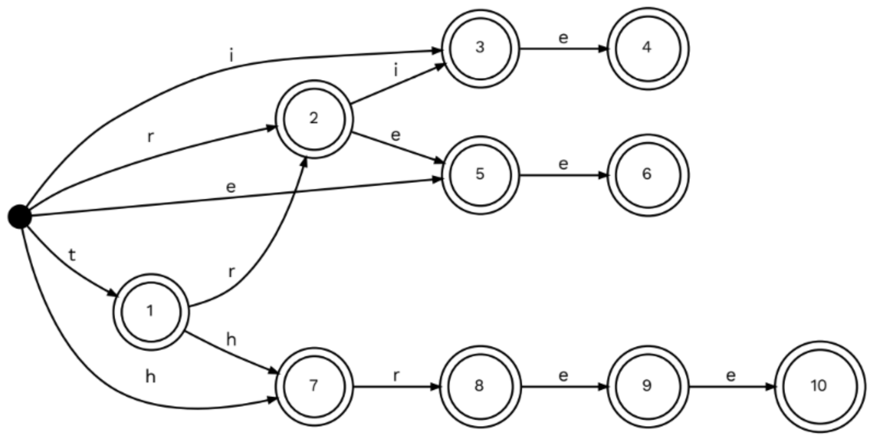
(d) In this step of the construction algorithm, a new transition **from the initial state to state 3** is made, labelled with i. There are now 13 transitions in total.

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Construction Algorithm of FOs



(e) In this step of the construction algorithm, a new transition **from the initial state to state 5** is made, labelled with e. There are now 14 transitions in total.



(f) In the final step of the construction algorithm, **all states are marked as final states**. The algorithm transformed the original trie to the factor oracle of $P = \{\texttt{trie, tree, three}\}$.

Interactive Demonstration of FOs

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

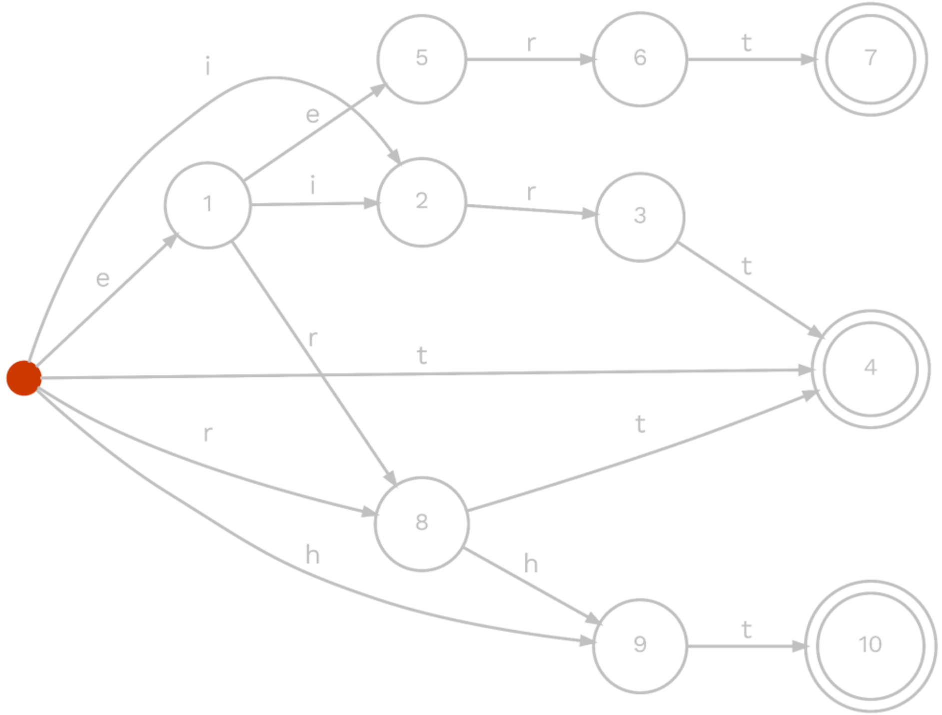
Applications of FOs

- **Used for pattern matching:**
  - Backwards Oracle Matching (BOM)
  - Set Backwards Oracle Matching (SBOM)

Set Backward Oracle Matching

two␣or␣three␣trees

(a) We attempted to input ␣ in the factor oracle, but the oracle rejects directly. We shift the window after the ␣ and continue our search.

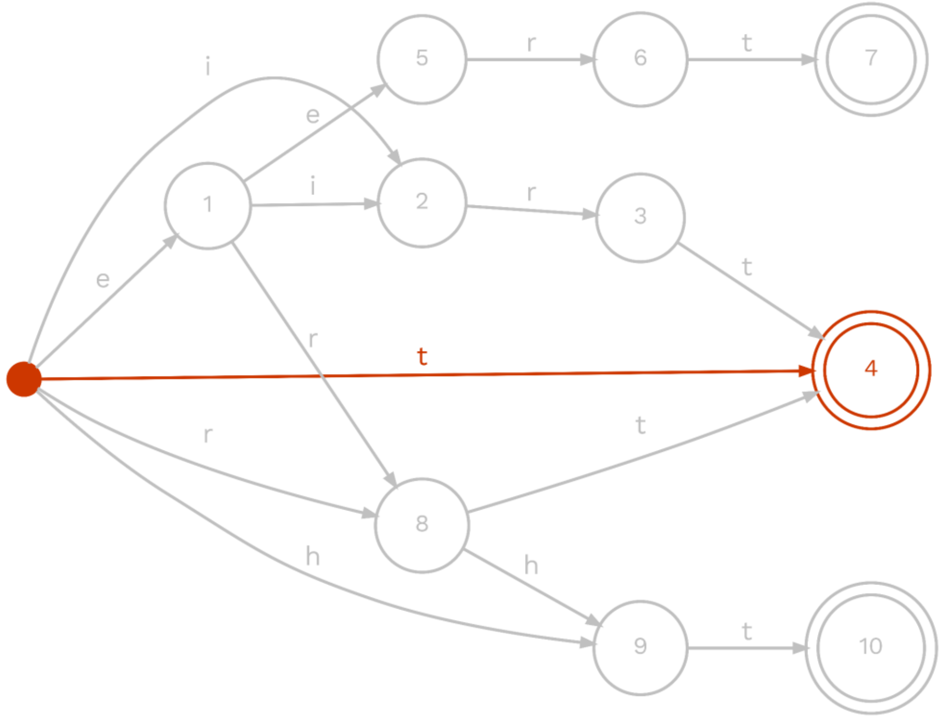


ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Set Backward Oracle Matching

two␣ or␣t hree␣trees

(b) We successfully input *t* in the factor oracle, but the oracle fails on ␣. We shift the window after the ␣ and continue our search.

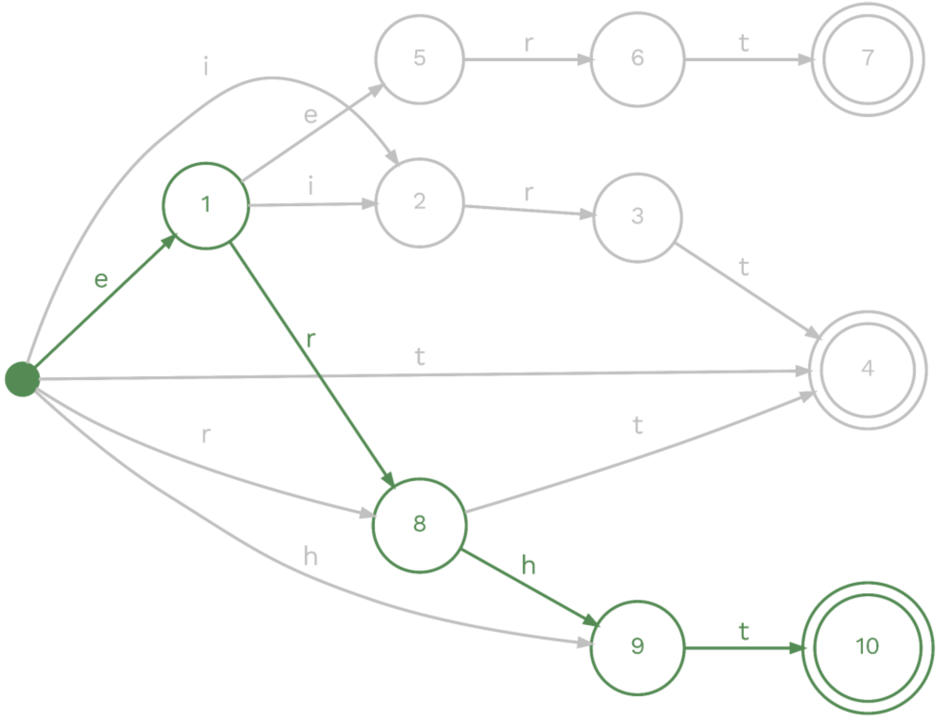


ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Set Backward Oracle Matching

two␣or␣three␣trees

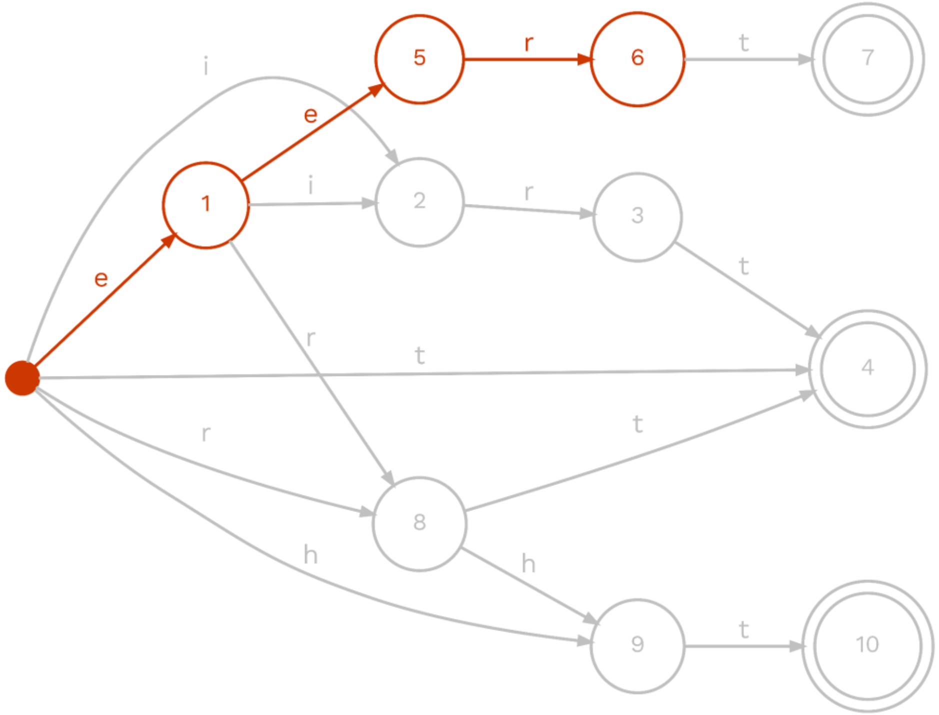
(c) We successfully input *e*, *r*, *h*, and *t* in the oracle. We reach accepting state 10, which is associated with the word *three*. We check for an occurrence of *three*, and shift the window by 1.



Set Backward Oracle Matching

two␣or␣three␣trees

(d) We successfully input e, e, and r in the oracle, but the oracle fail on the character h. We shift the window after the h and continue our search.

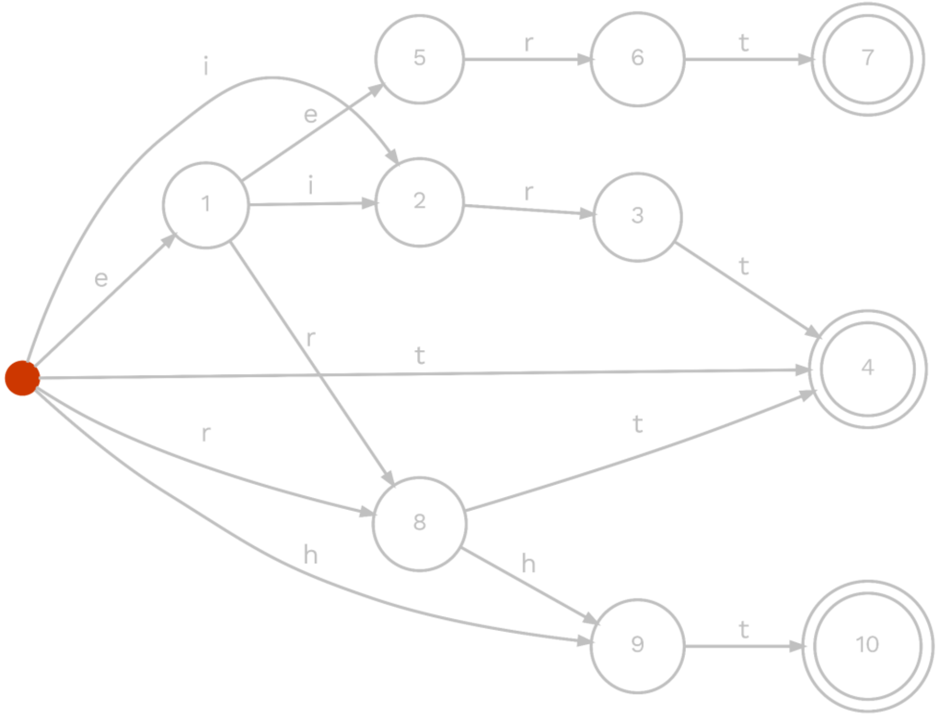


ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Set Backward Oracle Matching

two␣or␣th ree␣ trees

(e) We attempted to input ␣ in the factor oracle, but the oracle rejects directly. We shift the window after the ␣ and continue our search.

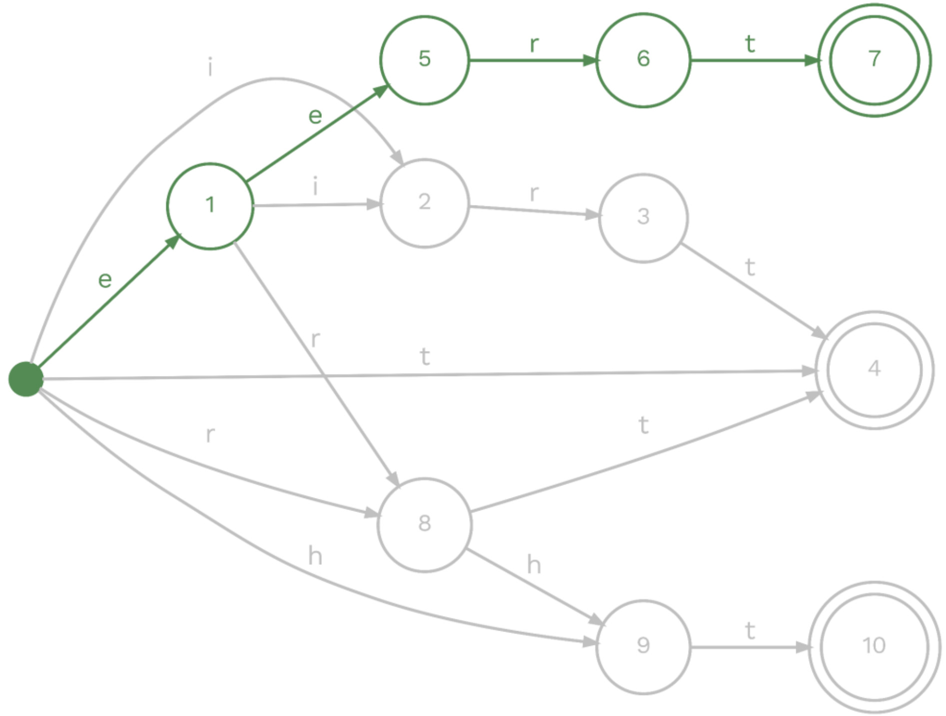


ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Set Backward Oracle Matching

two␣or␣three␣trees

(f) We input *e*, *e*, *r*, and *t* in the oracle. We reach accepting state 7, which is associated with the word *tree*. Our window contains *tree*.



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Applications of FSMs (2)

- **Anomaly detection on system call sequences:**
  - Could we use **factor oracles** for this?

- **Problems:**
  - Explosion in size
  - Narrowly-defined vs. broad behavior

```
S0;
while (..) {
      S1;
      if (...) S2;
      else S3;
      if (S4) ... ;
      else S2;
      S5;
}
S3;
S4;
```

$S_0S_1S_2$	$S_1S_2S_4$	$S_2S_4S_5$	$S_3S_4S_5$	$S_4S_5S_1$	$S_2S_5S_1$	$S_5S_1S_2$
$S_0S_1S_3$	$S_1S_3S_4$	$S_2S_4S_2$	$S_3S_4S_2$	$S_4S_5S_3$	$S_2S_5S_3$	$S_5S_1S_3$
$S_0S_3S_4$				$S_4S_2S_5$		$S_5S_3S_4$

Figure 1. An example program and associated trigrams. S0,...,S5 **denote system calls.**

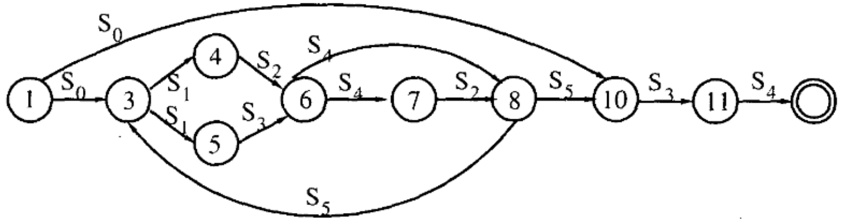


Figure 2. Automaton learnt by our algorithm for Example 1

Sekar, R., Bendre, M., Dhurjati, D., & Bollineni, P. (2000, May). A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001* (pp. 144-155). IEEE.

Outline of the lecture

- Preliminary knowledge:
  - Terminology + basics of automata theory
  - Tries, factor automata, and factor oracles

- **Our research at the AI lab:**
  - Hierarchical-alphabet automata (HAAs)
  - Hierarchical factor oracles (HFOs)

- Applications of our research:
  - Anomaly detection with the HFO

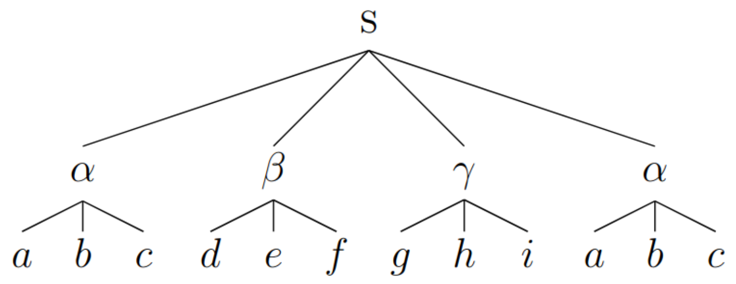
ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

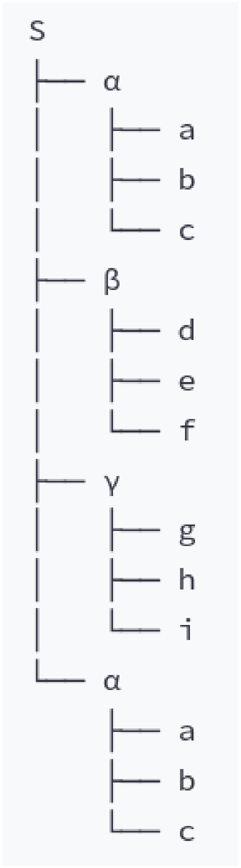
Research Goals

- **Solving previously-mentioned issues:**
  - Capturing broad, complex behavior with compact FSMs
- **Possible solution:**
  - Exploiting **hierarchical relationships** of symbols
- **How?**
  - Words to **hierarchical words** using hierarchical relationships
  - Factors to **hierarchical factors**
  - Finite state machines to **alphabet-hierarchical automata**

Hierarchical Words

- **Alphabet:** finite set of symbols
- **Hierarchical words:** concatenations of symbols in the form of a rooted tree
- **Hierarchical factors:** factors of hierarchically connected words for every level of the hierarchy





Hierarchical Words

- **Alphabet:** finite set of symbols
- **Hierarchical words:** concatenations of symbols in the form of a rooted tree
- **Hierarchical factors:** factors of hierarchically connected words for every level of the hierarchy

Hierarchical word:

```
S
├── α
│   ├── a
│   ├── b
│   └── c
├── β
│   ├── d
│   ├── e
│   └── f
├── γ
│   ├── g
│   ├── h
│   └── i
└── α
    ├── a
    ├── b
    └── c
```

Hierarchical factors:

```
S
├── β
│   ├── d
│   └── e
└── γ
    └── i
```

```
S
├── α
│   ├── a
│   ├── b
│   └── c
├── β
└── γ
    ├── g
    ├── h
    └── i
```

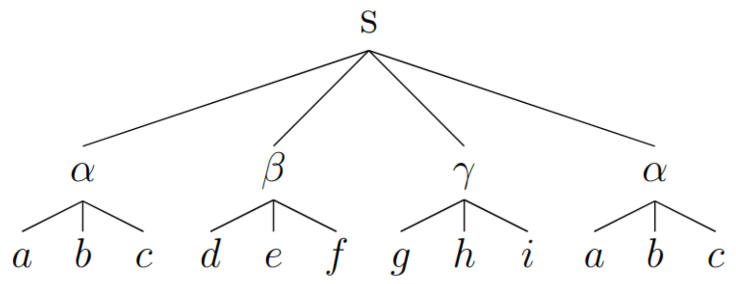
```
S
└── α
```

VUB ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Hierarchical Words

- **Where is hierarchy relevant?**
  - **Language:** words, sentences, paragraphs
  - **Music:** individual notes, chords
  - **Action:** actions into sequence of sub-actions

- **Hierarchy in cybersecurity?**
  - **Parent-child relationships between processes:** one process can spawn others

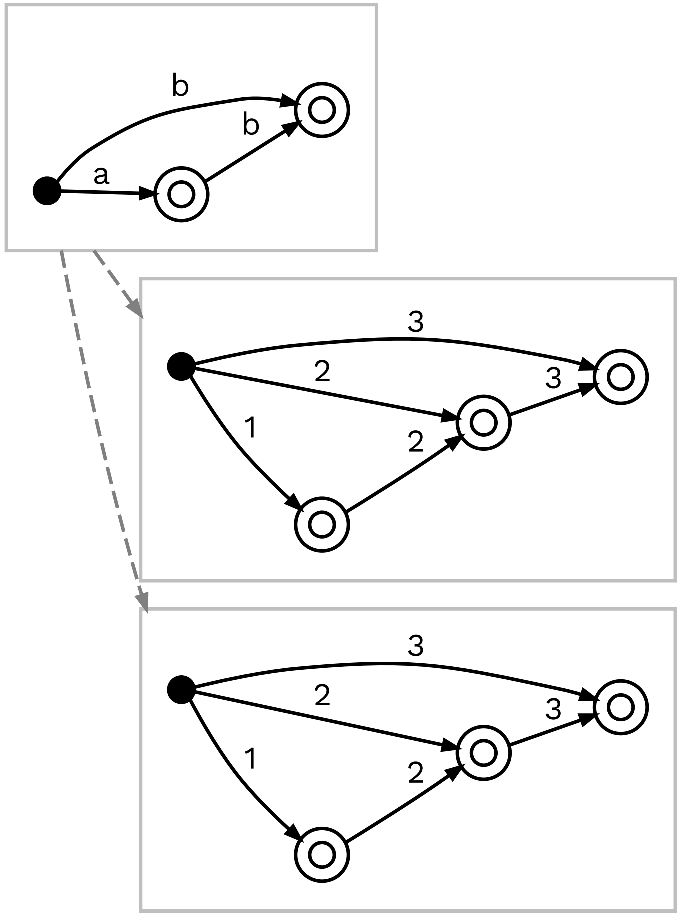



```
s
├── α
│   ├── a
│   ├── b
│   └── c
├── β
│   ├── d
│   ├── e
│   └── f
├── γ
│   ├── g
│   ├── h
│   └── i
└── α
    ├── a
    ├── b
    └── c
```

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

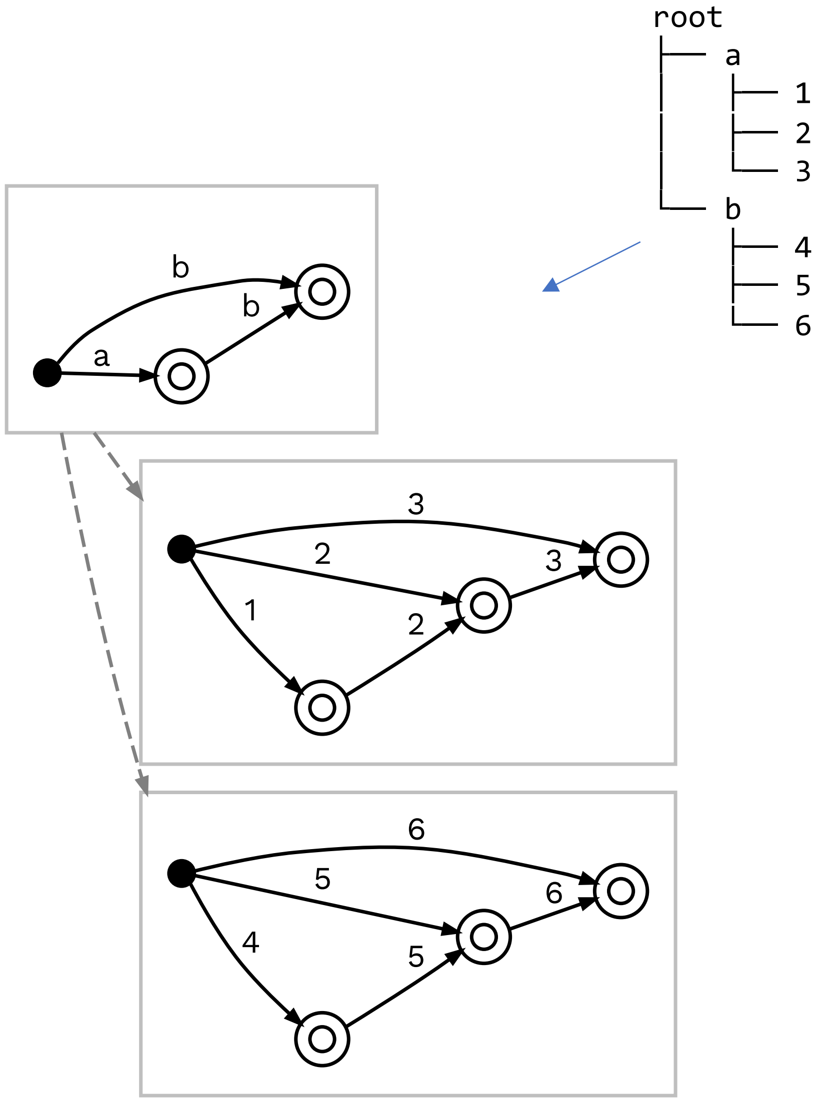
Hierarchical-Alphabet Automata

- **Intuitively:**
  - Hierarchical variant of regular FSMs
  - Has a **super-transition function *Δ*** that maps symbols of its alphabet in one level to (at most) one other HAA



Hierarchical Factor Oracles

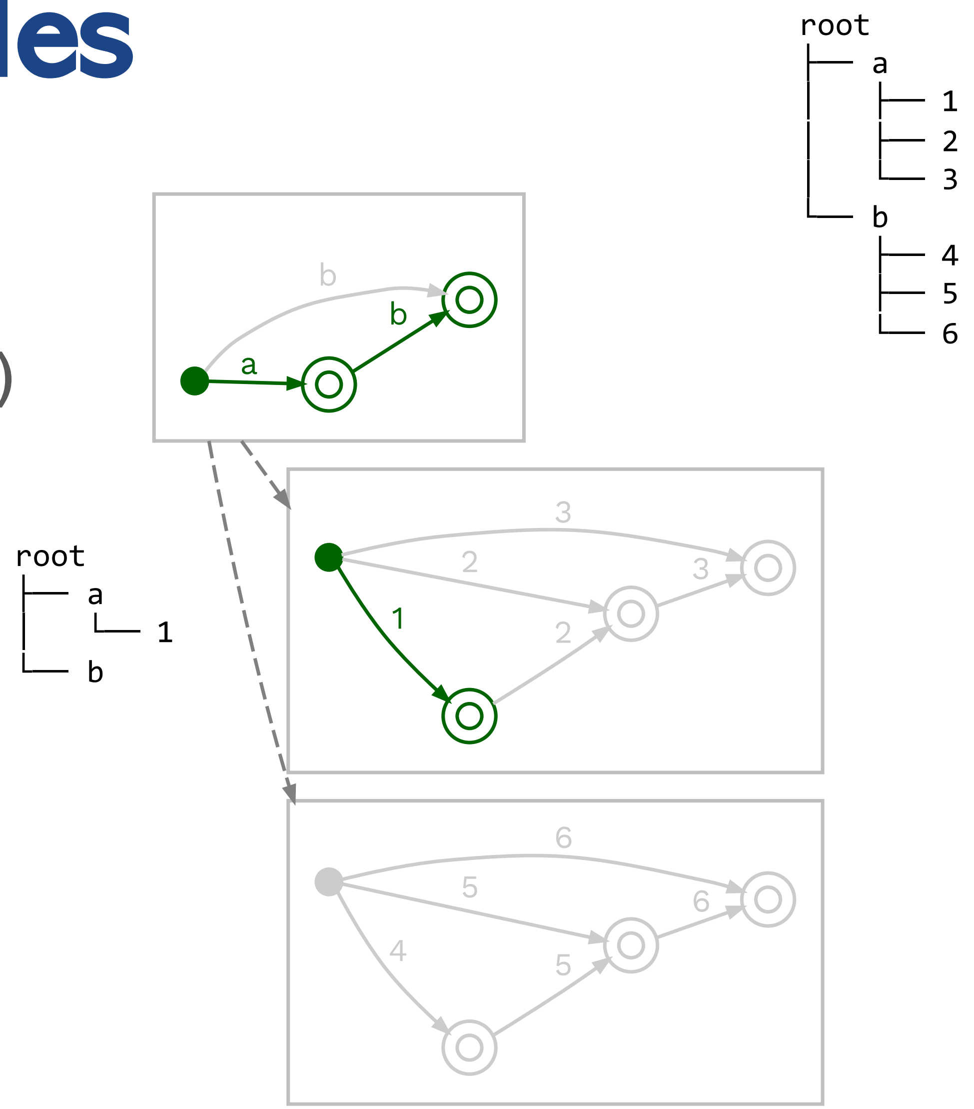
- **Hierarchical-alphabet variant of regular (flat) factor oracles:**
  - Built using a (set of) hierarchical word(s)
  - Accepts **hierarchical factors** of its set



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

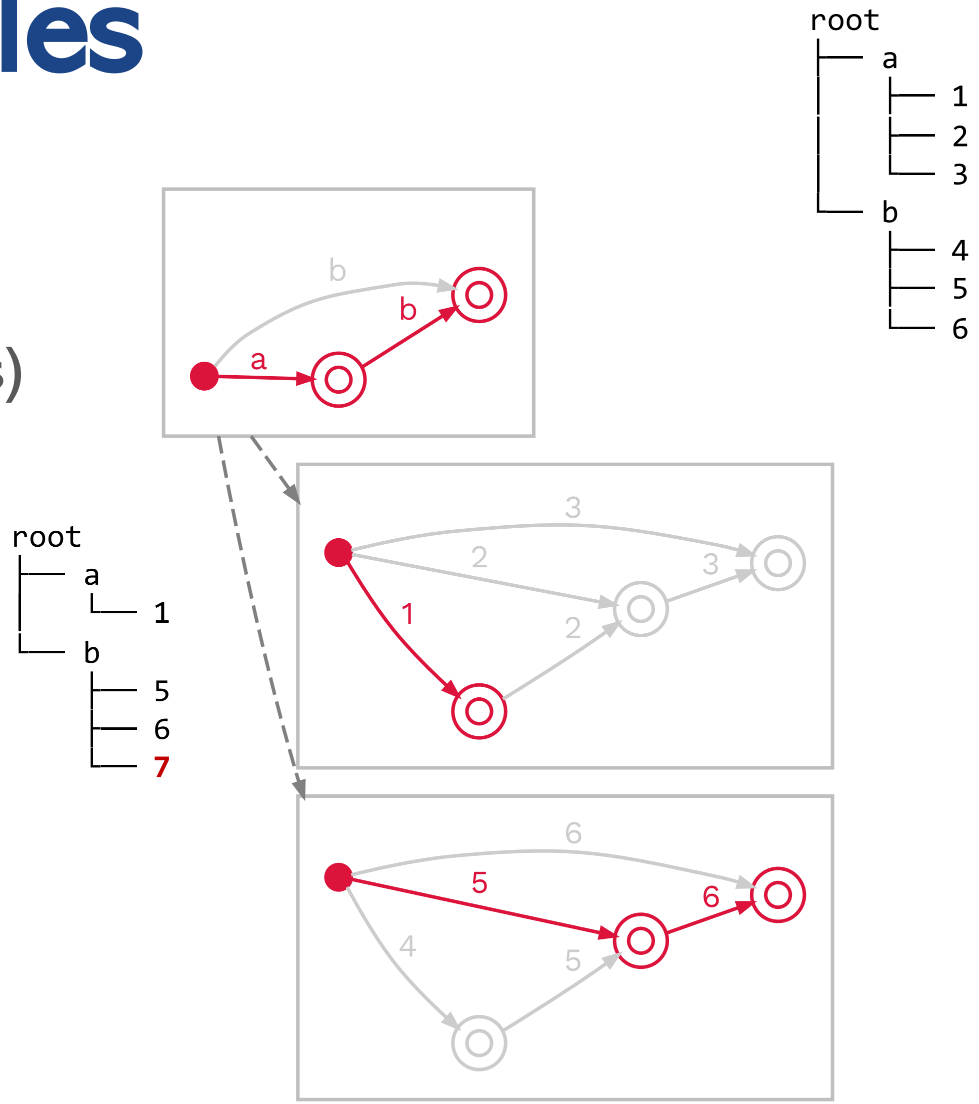
Hierarchical Factor Oracles

- **Hierarchical-alphabet variant of regular (flat) factor oracles:**
  - Built using a (set of) hierarchical word(s)
  - Accepts **hierarchical factors** of its set



Hierarchical Factor Oracles

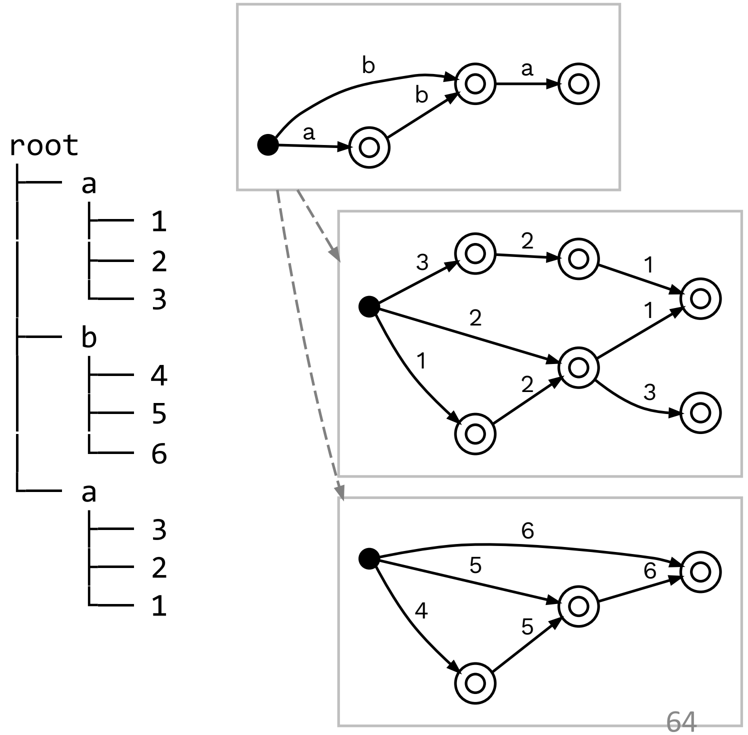
- **Hierarchical-alphabet variant of regular (flat) factor oracles:**
  - Built using a (set of) hierarchical word(s)
  - Accepts **hierarchical factors** of its set



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

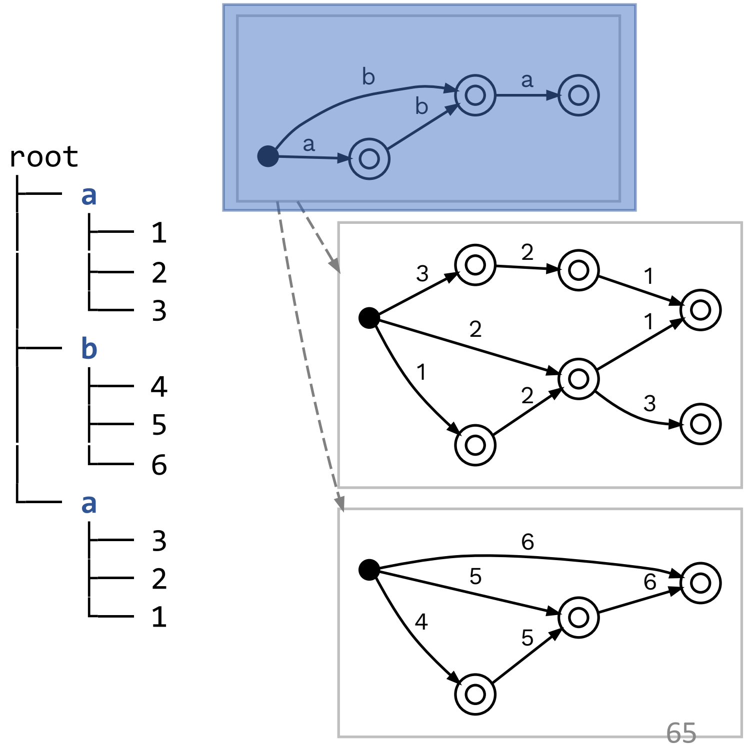
Construction Algorithm for HFOs

- **Hierarchical oracle for a set of hierarchical sequences:**
  - Perform a **breadth-first traversal** on each hierarchical sequence
  - During the traversal, build a **list of regular sequences**
  - **After traversal:** build regular FOs using regular sequences
  - Create **super-transitions from oracle to oracle**



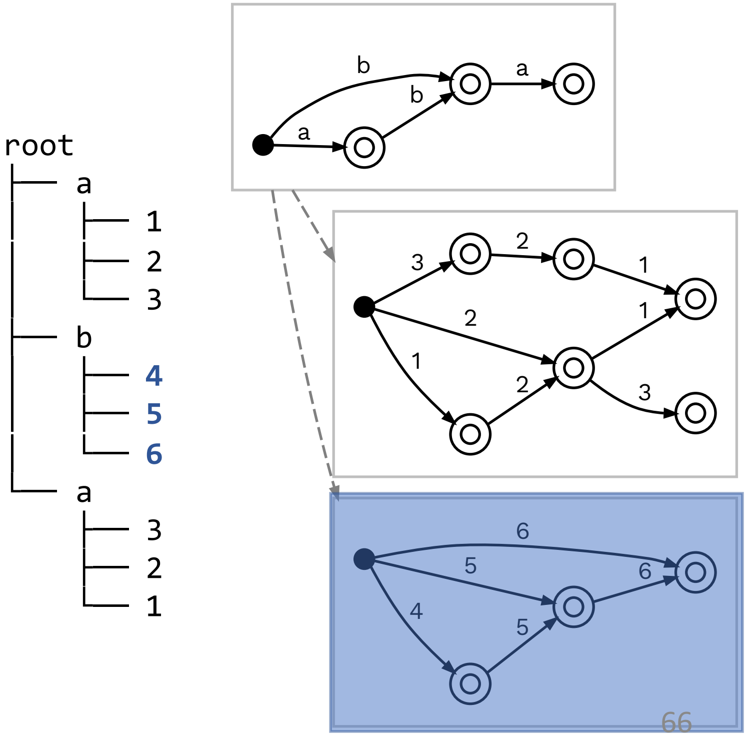
Construction Algorithm for HFOs

- **Hierarchical oracle for a set of hierarchical sequences:**
  - Perform a **breadth-first traversal** on each hierarchical sequence
  - During the traversal, build a **list of regular sequences**
  - **After traversal:** build regular FOs using regular sequences
  - Create **super-transitions from oracle to oracle**



Construction Algorithm for HFOs

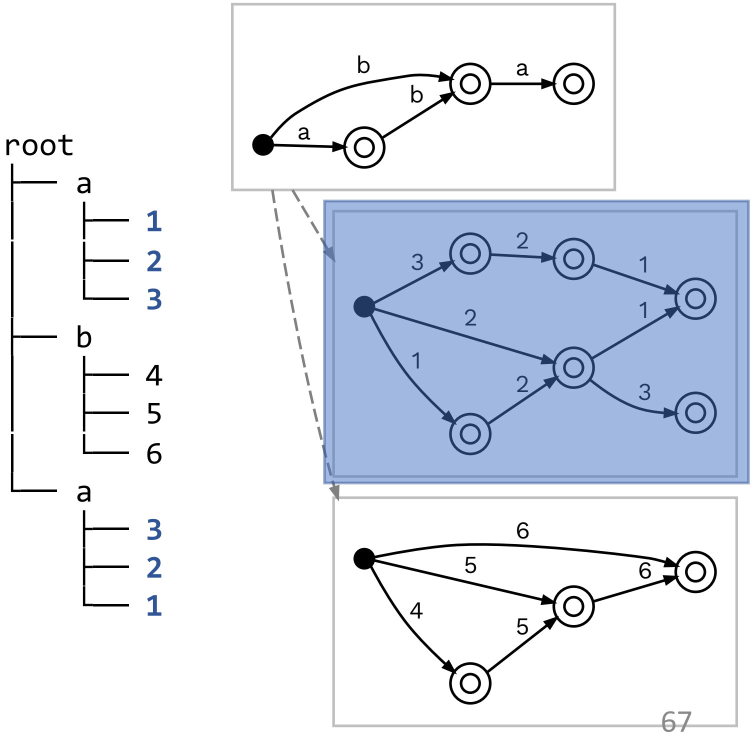
- **Hierarchical oracle for a set of hierarchical sequences:**
  - Perform a **breadth-first traversal** on each hierarchical sequence
  - During the traversal, build a **list of regular sequences**
  - **After traversal:** build regular FOs using regular sequences
  - Create **super-transitions from oracle to oracle**



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Construction Algorithm for HFOs

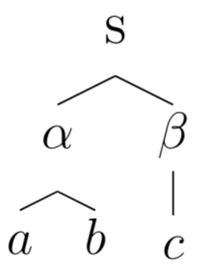
- **Hierarchical oracle for a set of hierarchical sequences:**
  - Perform a **breadth-first traversal** on each hierarchical sequence
  - During the traversal, build a **list of regular sequences**
  - **After traversal:** build regular FOs using regular sequences
  - Create **super-transitions from oracle to oracle**



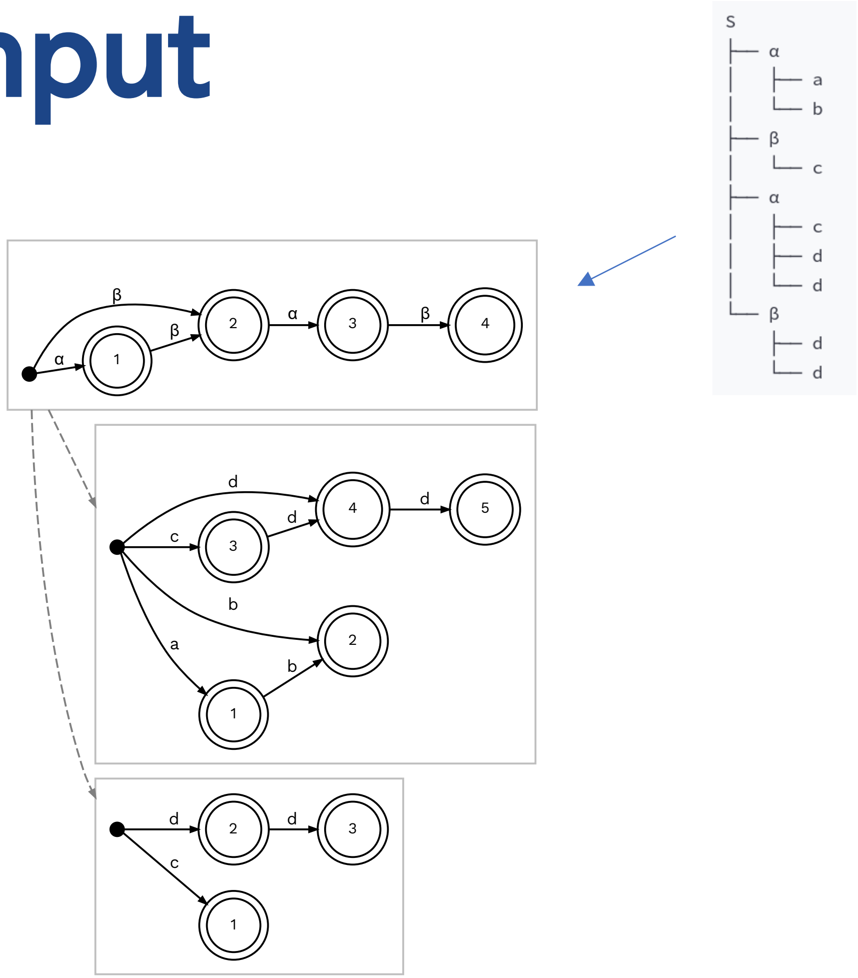
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Accepting or rejecting input

Accepting



with

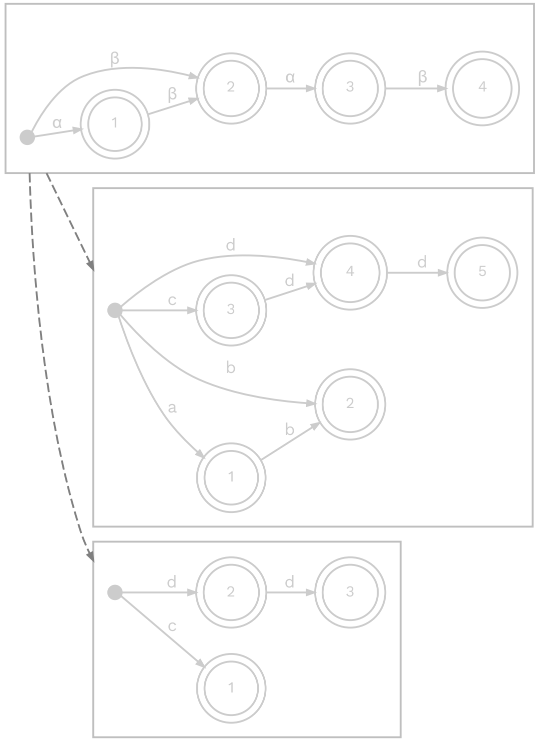


Accepting or rejecting input

```
S
├── α
│    └── a
│    └── b
└── β
     └── c
```

Stack = []

Current = None

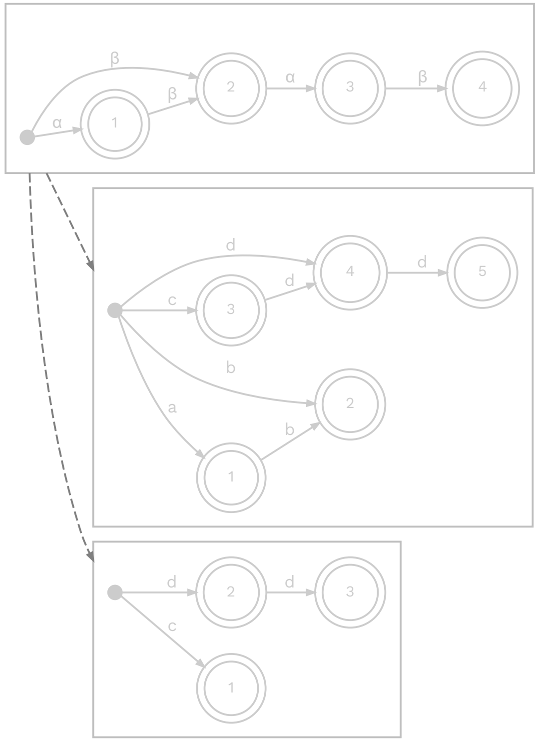


Accepting or rejecting input

```
S
├── α
│   └── a
│   └── b
└── β
    └── c
```

Stack = [β]

Current = None

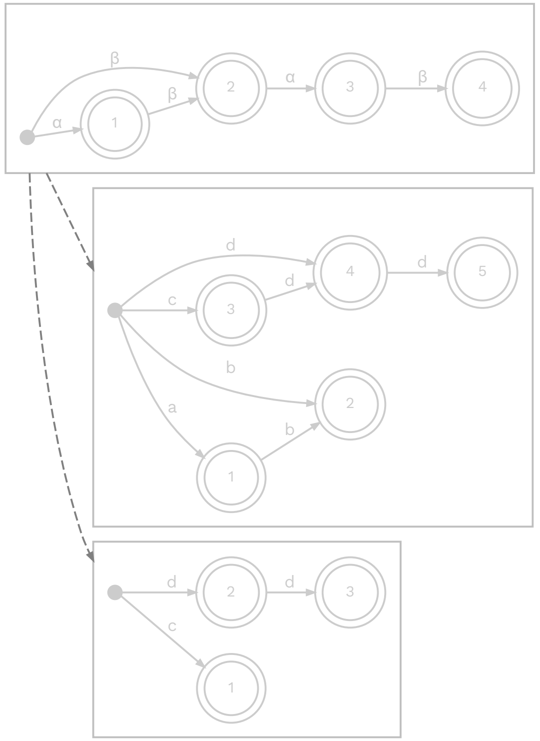


ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Accepting or rejecting input

```
S
├── α
│    └── a
│    └── b
└── β
     └── c
```

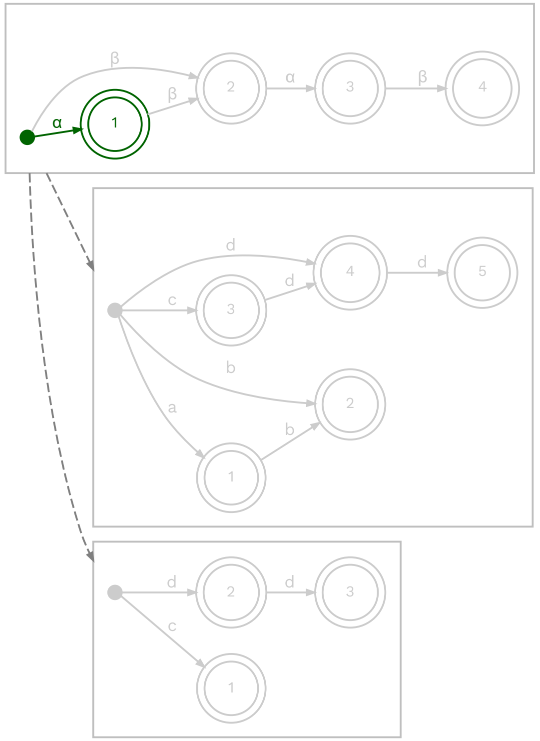
Stack = [α, β]
Current = None



Accepting or rejecting input

```
S
├── α
│   └── a
│   └── b
└── β
    └── c
```

Stack = [β]
Current = α



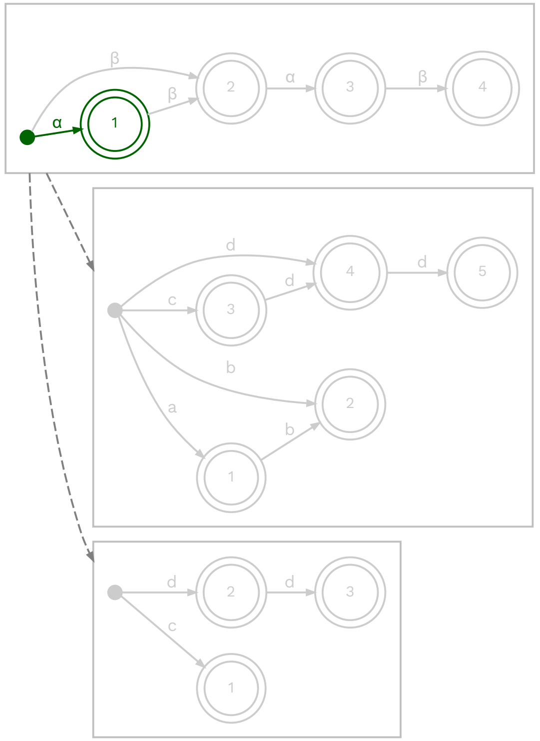
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Accepting or rejecting input

```
S
├── α
│   └── a
│   └── b
└── β
    └── c
```

Stack = [b, β]

Current = α



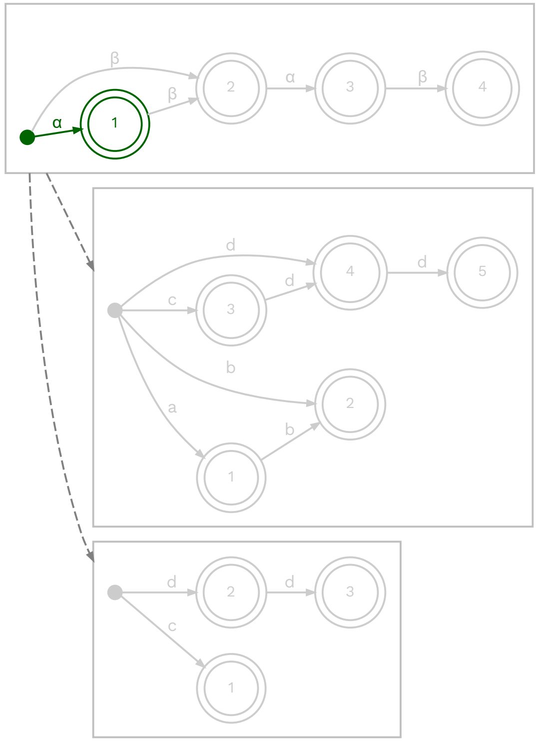
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Accepting or rejecting input

```
S
├── α
│    └── a
│    └── b
└── β
     └── c
```

Stack = [a, b, β]

Current = α



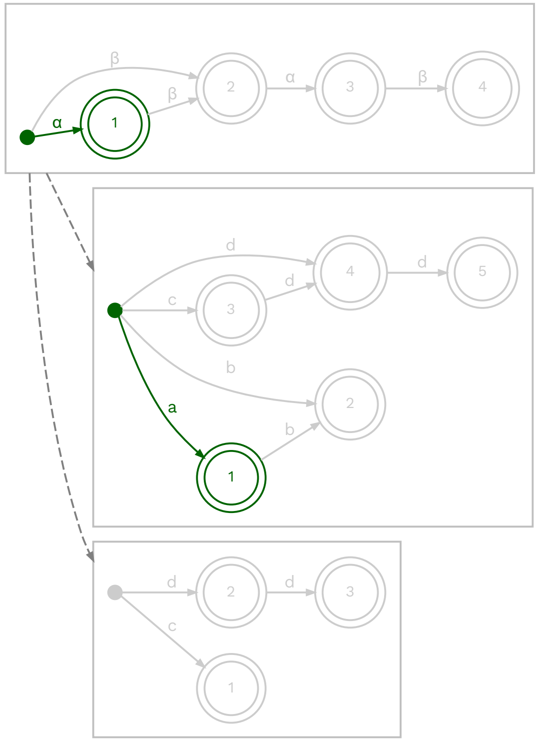
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Accepting or rejecting input

```
S
├── α
│   └── a
│   └── b
└── β
    └── c
```

Stack = [b, β]

Current = a

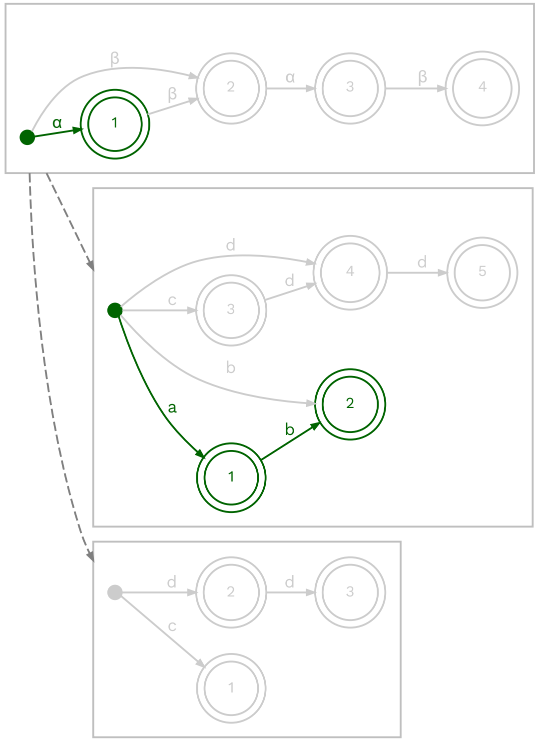


Accepting or rejecting input

```
S
├── α
│   └── a
│   └── b
└── β
    └── c
```

Stack = [β]

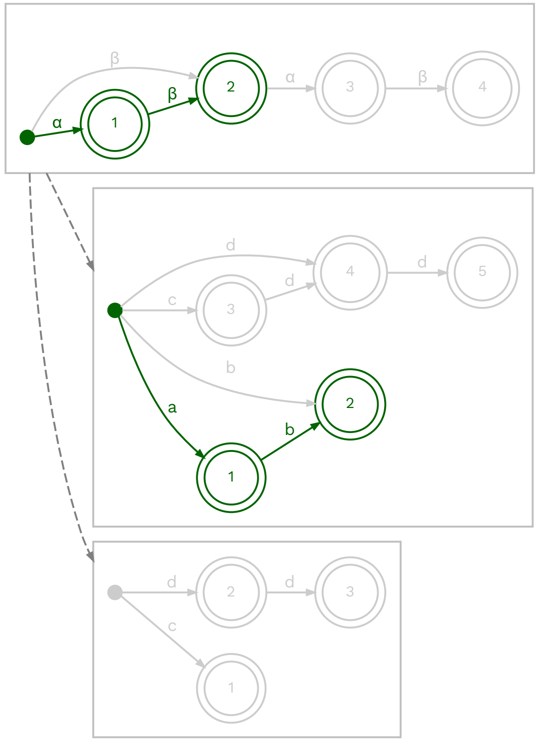
Current = b



Accepting or rejecting input

```
S
├── α
│   └── a
│   └── b
└── β
    └── c
```

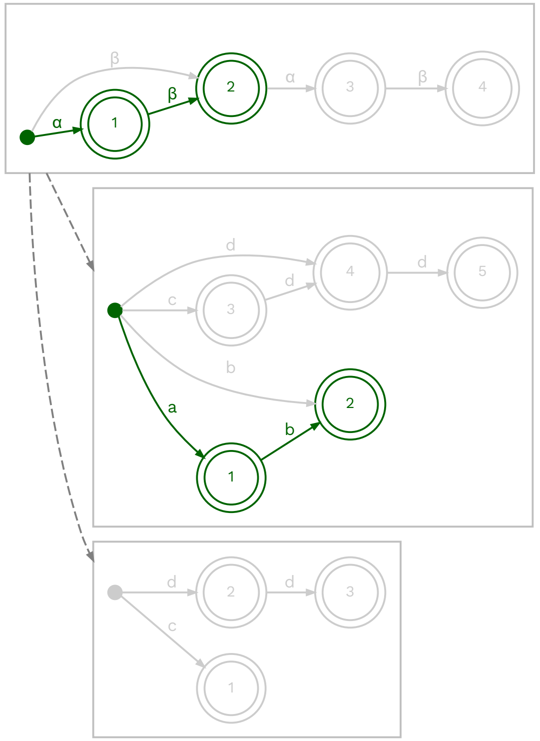
Stack = []
Current = β



Accepting or rejecting input

```
S
├── α
│    └── a
│    └── b
└── β
     └── c
```

Stack = [c]
Current = β

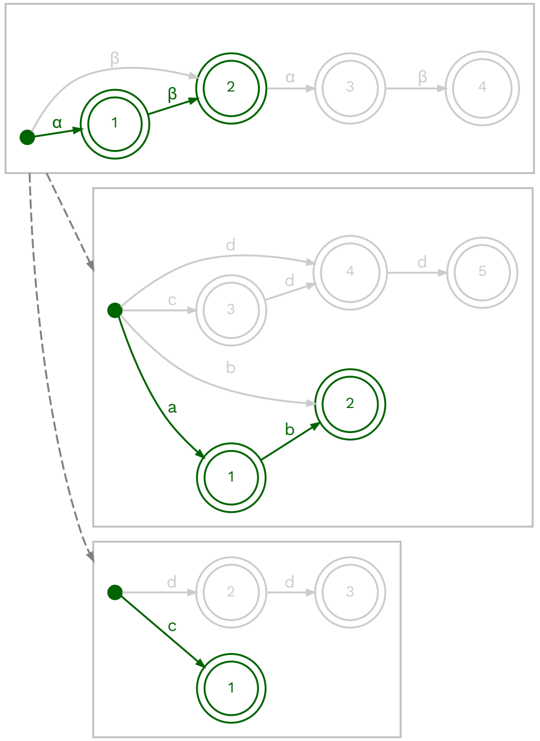


Accepting or rejecting input

```
S
├── α
│   └── a
│   └── b
└── β
    └── c
```

Stack = []

Current = c



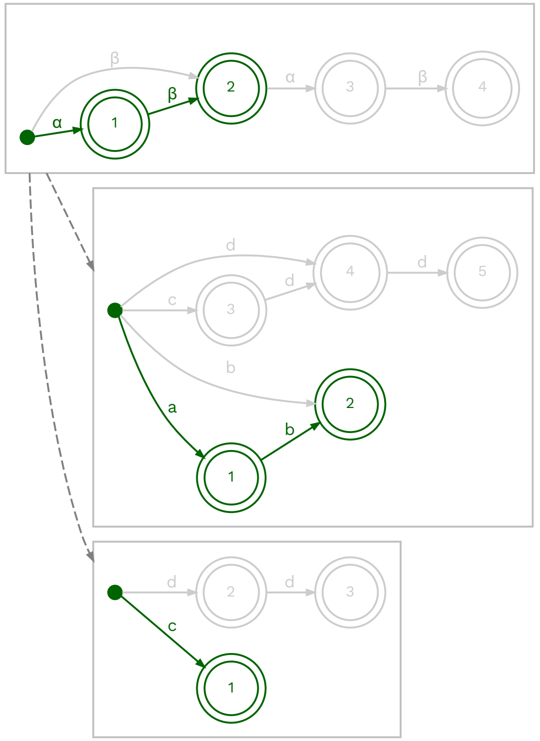
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Accepting or rejecting input

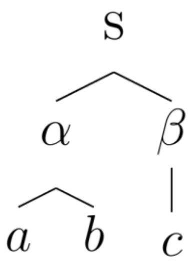
```
S
├── α
│    └── a
│    └── b
└── β
     └── c
```

Stack = []

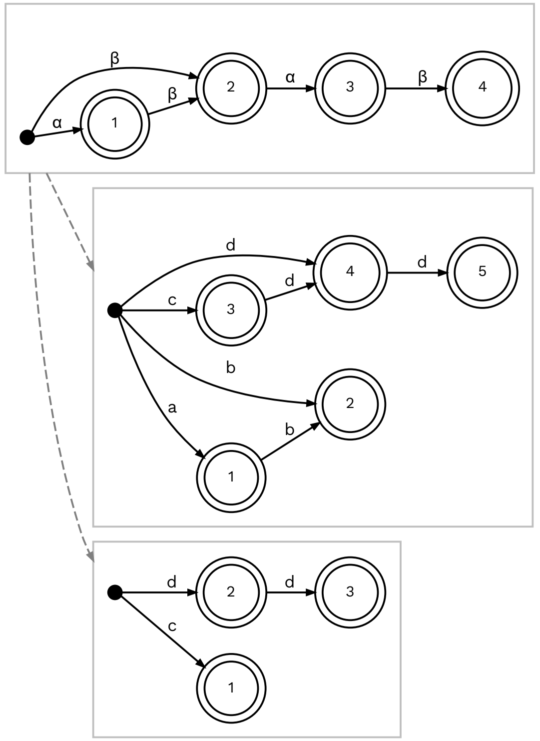
Current = None



Accepting or rejecting input



accepted by



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Interactive Demonstration of HFOs

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

Outline of the lecture

- **Preliminary knowledge:**
  - Terminology + basics of automata theory
  - Tries, factor automata, and factor oracles

- **Our research at the AI lab:**
  - Hierarchical-alphabet automata (HAAs)
  - Hierarchical factor oracles (HFOs)

- **Applications of our research:**
  - Anomaly detection with the HFO

VUB ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Anomaly Detection

- **What is it?**
  - Finding patterns in data that do not conform to expected behavior
  - **Relevant in lots of domains and applications, and real-life relevance!**

- **(Time) series anomaly detection:**
  - Detecting spikes, drops, out-of-place patterns, unexpected trends, …

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

HPM-based Anomaly Detection

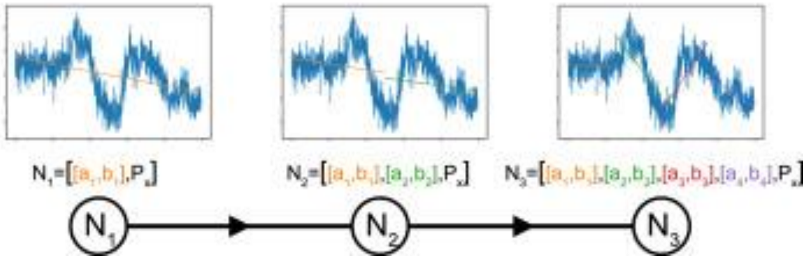
- **Main focus:** anomaly detection in production environments
  - Needs to be **lightweight** due to constraints in resources
  - We should never forget **rarely-occurring events**
  - We should be able to adapt to **new behavior**
  - It should be **general** enough to work on different time series
  - It should **recognize non-strictly periodic events**
  - Should have **a minimum amount of false positives**

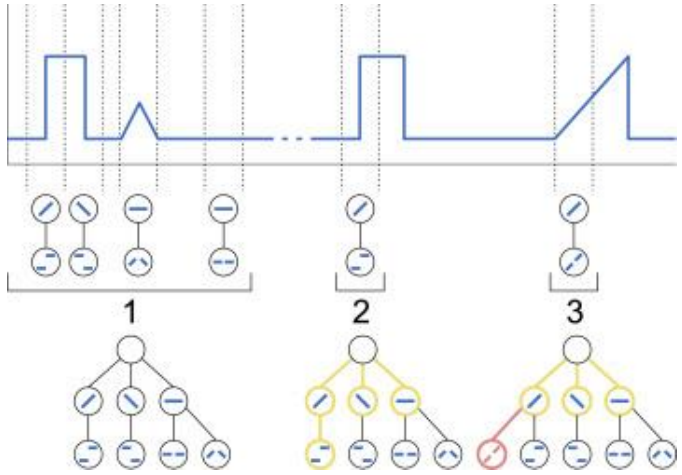
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Van Onsem, M., De Paepe, D., Hautte, S. V., Bonte, P., Ledoux, V., Lejon, A., ... & Van Hoecke, S. (2022). Hierarchical pattern matching for anomaly detection in time series. *Computer Communications*, *193*, 75-81.

HPM-based Anomaly Detection

- **Key idea:** extract hierarchical fingerprints from time series
  - Store all seen fingerprints in a tree structure
  - New fingerprints are marked as possible anomalies

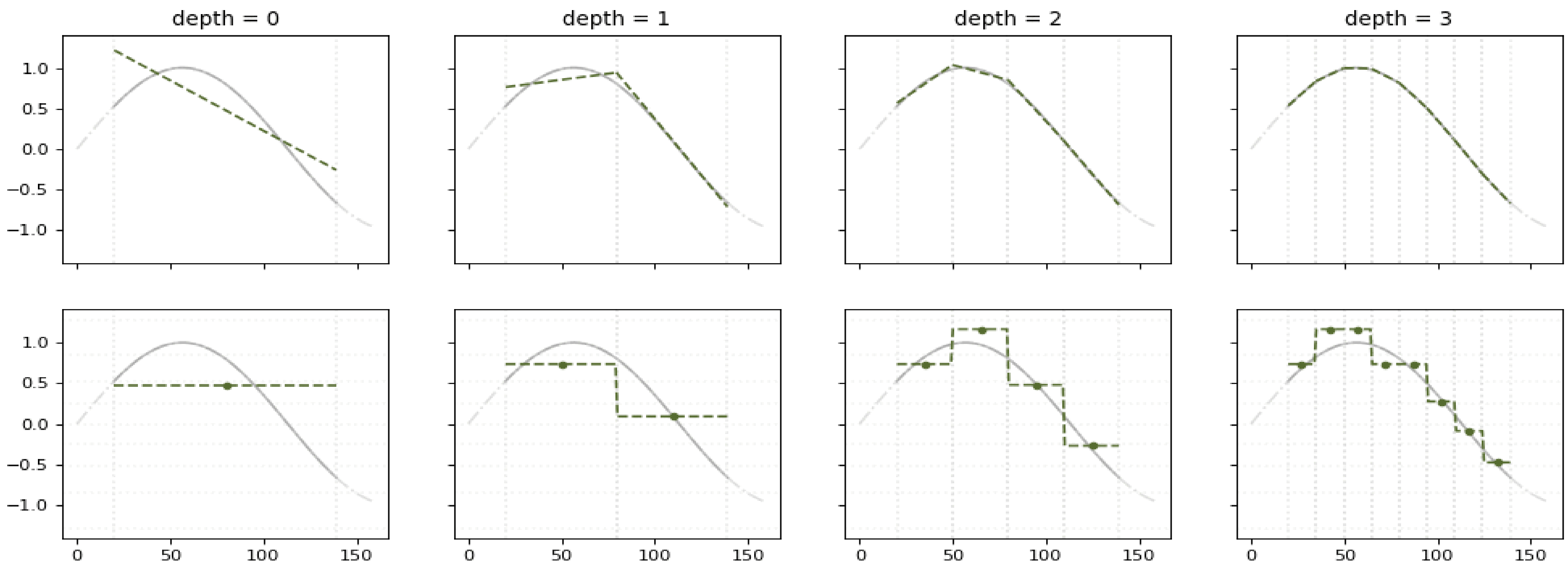




ARTIFICIAL INTELLIGENCE RESEARCH GROUP

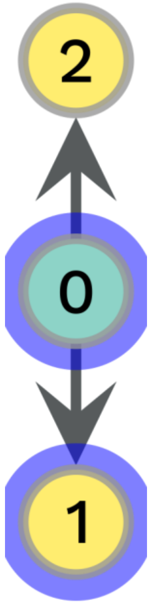
Van Onsem, M., De Paepe, D., Hautte, S. V., Bonte, P., Ledoux, V., Lejon, A., ... & Van Hoecke, S. (2022). Hierarchical pattern matching for anomaly detection in time series. *Computer Communications*, *193*, 75-81.

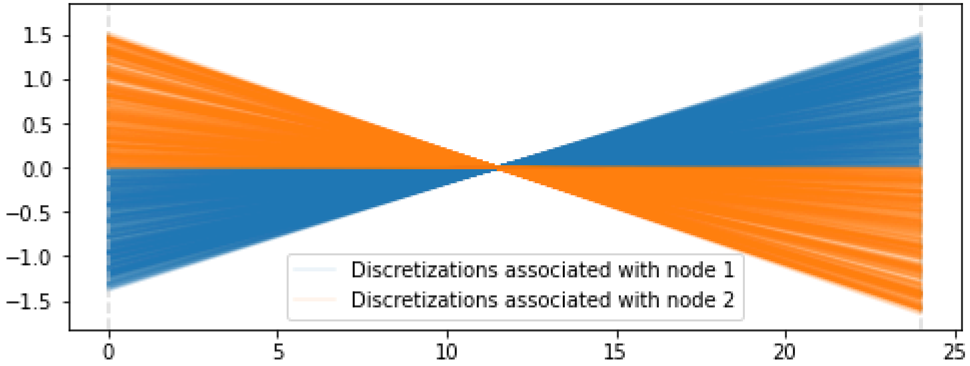
HPM-based Anomaly Detection



ARTIFICIAL INTELLIGENCE RESEARCH GROUP

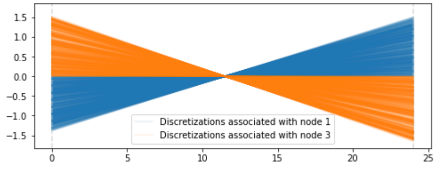
HPM-based Anomaly Detection

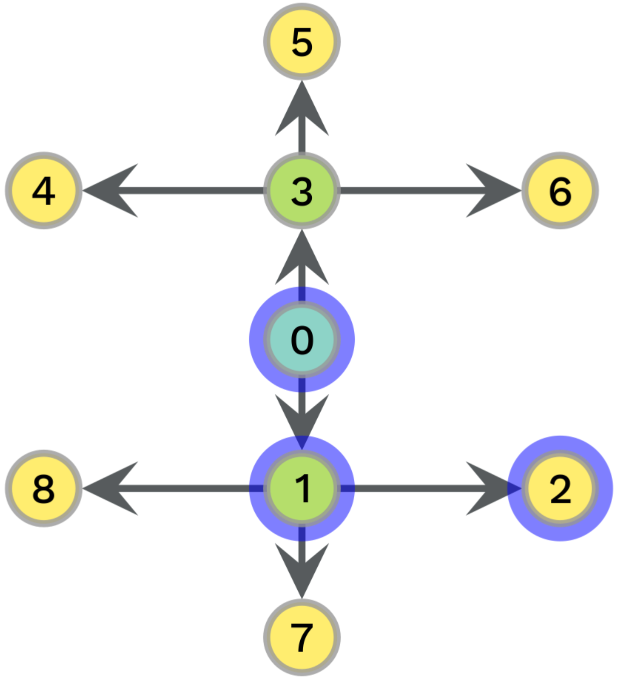


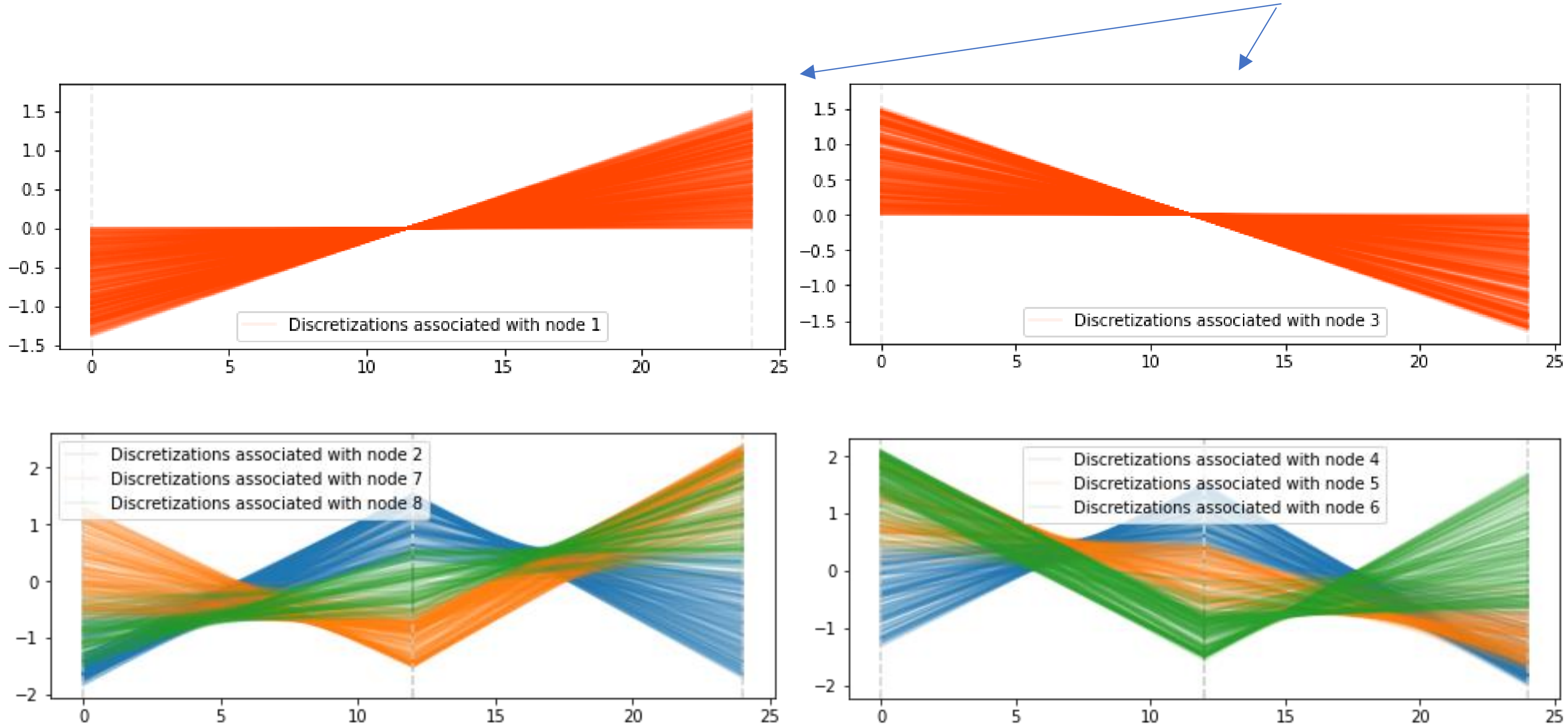


ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

HPM-based Anomaly Detection

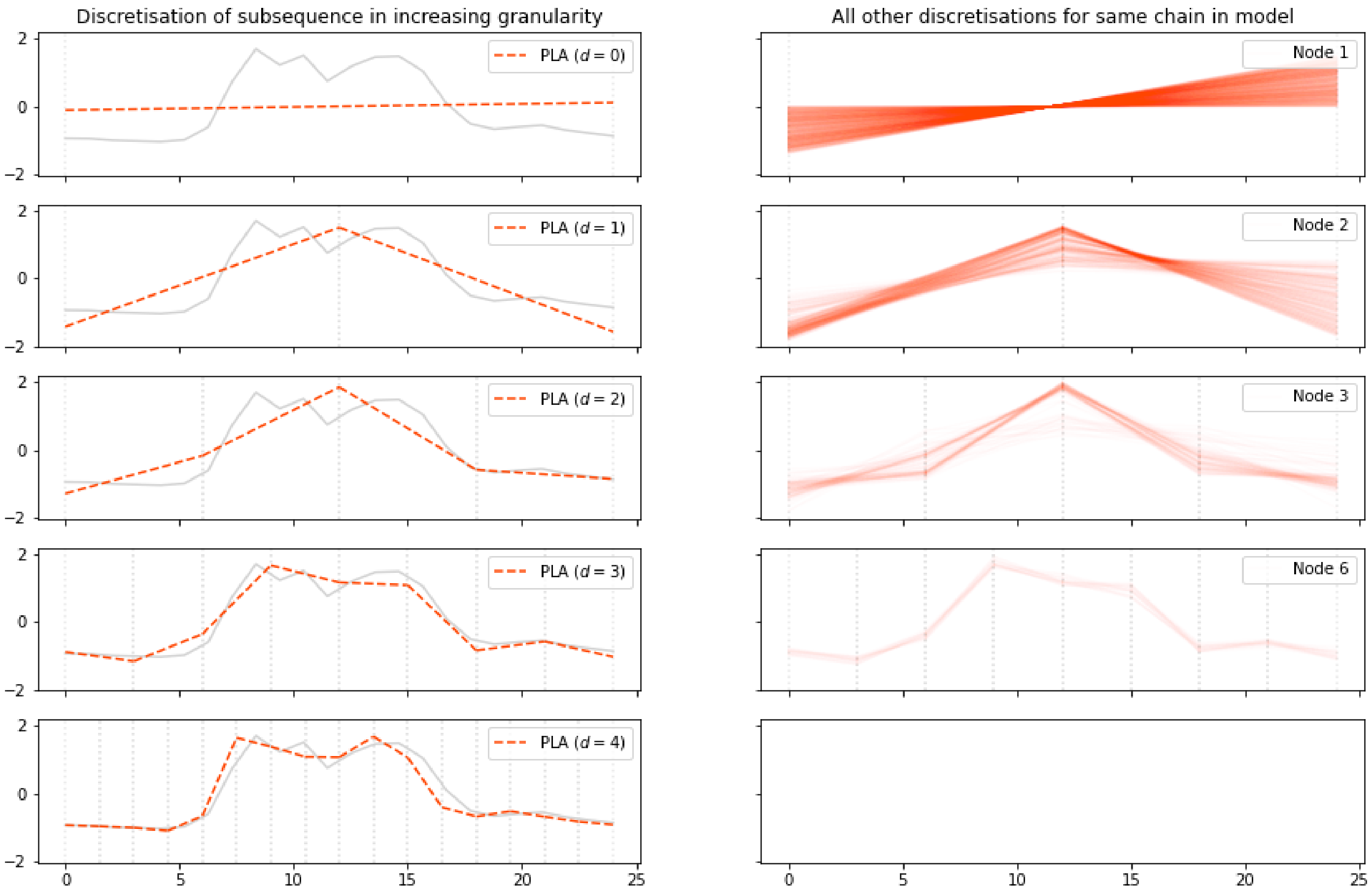


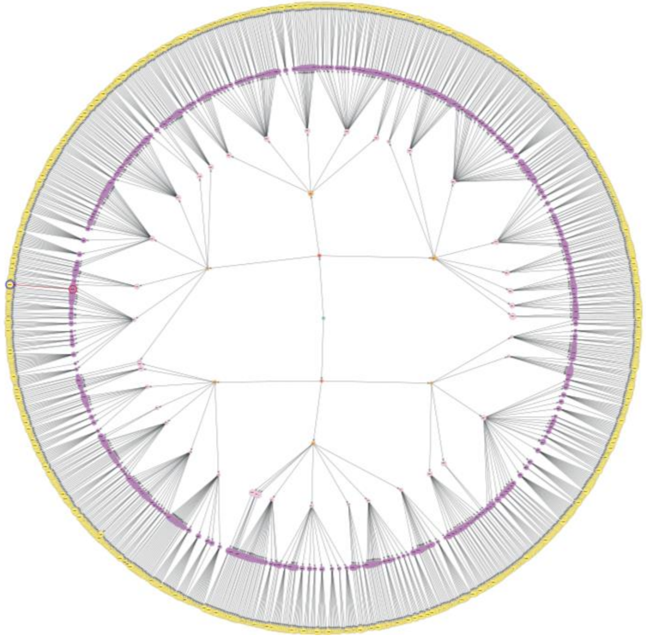




VUB ARTIFICIAL INTELLIGENCE RESEARCH GROUP

HPM-based Anomaly Detection





ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Interactive Demonstration of HPM

HPM-based Anomaly Detection

- **Disadvantages of basic algorithm:**
  - **Everything is anomalous if you go fine-grained enough**
  - **No temporal anomalies:** only new patterns are possible anomalies
  - **Lack of context:** have we seen a *factor* of fingerprints before?

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Extending HPM with HFOs

- **Possible approach:**
  1. **Segmenting:** extract subsequences using a sliding window
  2. **Discretizing:** extract fingerprints from above-obtained subsequences
  3. **Learning:** represent consecutive fingerprints as hierarchical words (and HFO)
  4. **Anomaly detection:** calculate score based on largest accepted HFO input

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

Extending HPM with HFOs

- **Segmenting:**

```
                           requests
2022-07-31 22:00:00+00:00 29309
2022-07-31 23:00:00+00:00 29977
2022-08-01 00:00:00+00:00 29748
2022-08-01 01:00:00+00:00 28414
2022-08-01 02:00:00+00:00 27871
...                           ...
2022-10-14 17:00:00+00:00 41931
2022-10-14 18:00:00+00:00 41215
2022-10-14 19:00:00+00:00 38529
2022-10-14 20:00:00+00:00 37104
2022-10-14 21:00:00+00:00 34412
```

```
[[29309 29977 29748 ... 40237 36560 34145]
 [29977 29748 28414 ... 36560 34145 31998]
 [29748 28414 27871 ... 34145 31998 31041]
 ...
 [37564 34118 29764 ... 41931 41215 38529]
 [34118 29764 29600 ... 41215 38529 37104]
 [29764 29600 29485 ... 38529 37104 34412]]
```

Extending HPM with HFOs

- **Discretizing:**

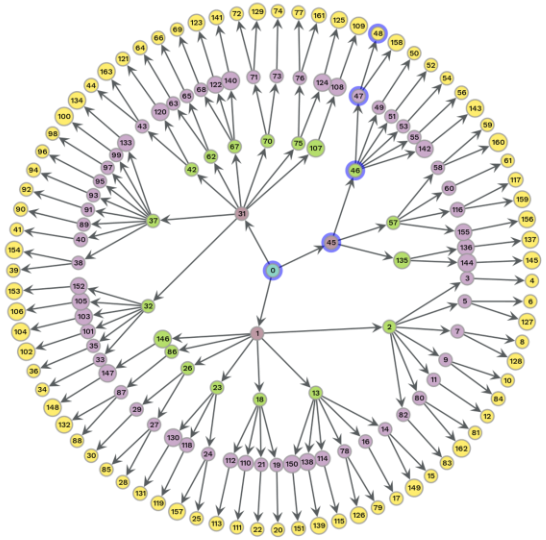
```
[[29309 29977 29748 ... 40237 36560 34145]
 [29977 29748 28414 ... 36560 34145 31998]
 [29748 28414 27871 ... 34145 31998 31041]
 ...
 [37564 34118 29764 ... 41931 41215 38529]
 [34118 29764 29600 ... 41215 38529 37104]
 [29764 29600 29485 ... 38529 37104 34412]]
```

```
[[Node(depth, slope, inter, …), Node(...), …]
 [Node(depth, slope, inter, …), Node(...), …]
 [Node(depth, slope, inter, …), Node(...), …]
 ...
 [Node(depth, slope, inter, …), Node(...), …]
 [Node(depth, slope, inter, …), Node(...), …]
 [Node(depth, slope, inter, …), Node(...), …]]
```

Extending HPM with HFOs

- **Learning:**

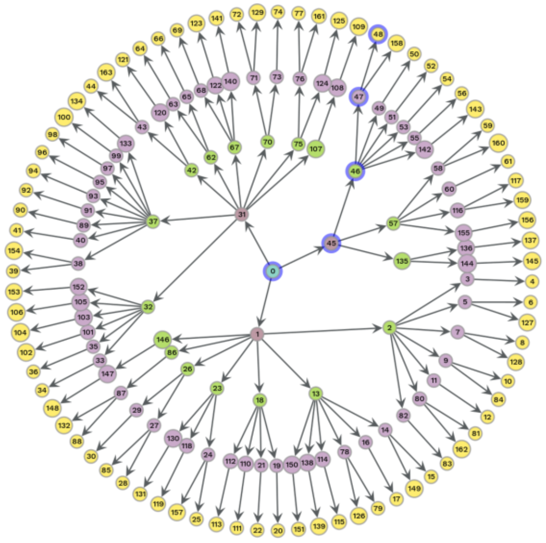
```
[[Node(depth, slope, inter), Node(...), …]
 [Node(depth, slope, inter), Node(...), …]
 [Node(depth, slope, inter), Node(...), …]
 ...
 [Node(depth, slope, inter), Node(...), …]
 [Node(depth, slope, inter), Node(...), …]
 [Node(depth, slope, inter), Node(...), …]]
```

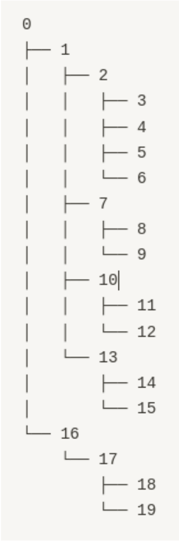


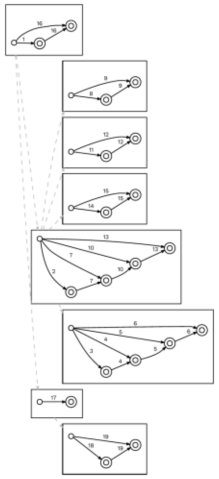
ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Extending HPM with HFOs

- **Learning:**







ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Extending HPM with HFOs

- **Advantages over HPM:**
  - **Not everything is anomalous:** search finds largest accepting factor
  - **Temporal anomalies:** old patterns are possibly anomalous
  - **Lack of context:** checks for *factors* of fingerprints

- **Advantages over HPM (extended with other state methods):**
  - **Compactness:** exploiting hierarchy of discretization for compactness
  - **Hierarchy:** takes hierarchy of discretization into account

VUB ARTIFICIAL INTELLIGENCE RESEARCH GROUP

Interactive Demonstration of Algorithm